Knowledge Acquisition and Management for Driving a Decision Support System

- BI Self-X -

By

Vlad Nicolicin Georgescu

A THESIS SUBMITTED TO POLYTECHNIQUE SCHOOL OF NANTES UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Jury:

Reviewers:	Jean-Marc PETIT	Professeur	INSA Lyon
	Jérôme DARMONT	Professeur	Université Lyon 2
President:	Gilles VENTURINI	Professeur	Polytech' Tours
Supervisors:	Henri BRIAND	Professeur Emérite	Polytech' Nantes
	Remi LEHN	Maître de conférences	Polytech' Nantes
Examiner	Pascale KUNTZ	Professeur	Polytech' Nantes
Guest	Vincent BENATIER	Gérant	SP2 Solutions

Abstract/Résumé

Abstract

This thesis combines three major research domains: (i) the management of information systems, specifically Decision Support Systems and Data Warehouses, (ii) autonomic task management using Autonomic Computing and (iii) the transformation and modeling of knowledge by adopting Web Semantic technologies and Ontologies.

From the principle of: you can't manage what you don't measure, decision support systems lack a proper awareness of their environment. Unlike operational systems, defining performance for DSS is a subjective task, as it is based on the quality of the provided service and the satisfaction of the users. Quantifying the performances of a DSS should not only take into account technical indicators, but also quality of service measures expressed using service license agreements and objectives. Autonomic Computing is a solution aimed at helping human IT experts with managing complex IT systems. Especially with DSS, many of the low level and repetitive tasks (such as parameter configuration) are performed by human experts. This means that they have less time left to focus on high level tasks such as business objective implementations. The core of the autonomic computing model is the autonomic computing manager, an intelligent four phase loop based on a central knowledge base. Defining this knowledge base, especially in a decisional environment is a challenging task. Knowledge sources vary from readme documents to human experience, many in non-structured forms. There is a need to integrate all this knowledge and make it useful for both human and machine. A solution to such integration is the web semantics technologies with ontologies. Ontologies permit the definition of all the concepts in the domain, and all the relation between these concepts.

In this context the two main issues are addressed: (i) the integration of the DSS management knowledge (structured and non-structured) into a unified knowledge base and (ii) the usage of the integrated knowledge base in a semi-autonomic manner (both by human and machine) with the purpose of improving the quality of the offered services to users.

The principal contributions of the thesis are:

(a) The elaboration of an ontology model of the DSS and its management policies, which includes architectures, parameters, technical performances, subjective performances (QoS), best practices, known issues and service levels (SLA/O)

(b) The elaboration of an autonomic computing adoption model which provides the DSS with self management functions: configuration, healing and optimization. These functions are elaborated around the improvement of the service levels and the quality of service. The autonomic computing manager loops are described with regards to the specifications of data warehouses as the core of DSS, specifically the hierarchal organization and the non continuous utilization patterns.

(c) The development of BI Self-X, composed of three modules each in charge of a self management function. The results obtained with this approach have proven that enterprises

using BI Self-X with their DSS have increased performance and service levels while decreasing the costs and time in the implementation and sustainability of their data warehouses.

Keywords: Decision Support System, Data Warehouse, Ontology, Autonomic Computing, Service Level Agreements, Quality of Service

Résumé

Les travaux de cette thèse combinent trois domaines de recherche : (i) la gestion des systèmes d'information, plus précisément les Systèmes d'Information Décisionnels (SID) et les entrepôts de données, (ii) la gestion autonomique avec le Calcul Autonomique et (iii) l'intégration des connaissances avec les technologies sémantiques et les ontologies.

Le principe fondamental de la gestion d'un système d'information est : ce qui ne se mesure pas, ne se gère pas. Contrairement aux systèmes opérationnels, définir des mesures pour un SID est une tâche subjective, basée sur les niveaux de service et la satisfaction d'utilisateurs. Actuellement, la gestion des entrepôts de données se focalise uniquement sur les aspects techniques, sans tenir compte des mesures de qualité de service décrites par les accords de niveau de service. Par conséquent, les experts perdent leur temps à traiter des tâches de bas niveau (techniques) au détriment des tâches de haut-niveau (satisfaction des objectifs métiers). Dans ce cadre de gestion, le Calcul Autonomique est une solution qui vise à les aider à automatiser certaines de leurs tâches. Il est décrit par un gestionnaire autonomique qui implémente une boucle intelligente avec quatre phases centrées sur une base de connaissances commune. Les sources de connaissances varient de documents techniques à l'expérience humaine et beaucoup sont dans des formes non-structurées. C'est pourquoi il y a un fort besoin d'intégration de ces connaissances pour les rendre utilisables par les experts et par les ordinateurs. Dans cette problématique, le domaine de l'ingénierie des connaissances propose les ontologies comme formalisme de représentation des connaissances. Une ontologie permet la définition des concepts d'un domaine et les relations qui existent entre ces concepts.

Dans ce contexte, la thèse adresse deux problématiques : (i) l'intégration des connaissances (structurées et non-structurées) pour la mise en place et maintenance des entrepôts de données en unifiant la représentation de leurs règles de gestion sous la forme d'une ontologie et (ii) l'utilisation de cette ontologie avec le Calcul Autonomique en tenant compte des particularités des SIDs. Les apports principaux de cette thèse sont :

(a) L'élaboration d'une ontologie qui modélise le SID et sa gestion, comprenant donc : l'architecture des entrepôts de données, les paramètres de configuration, les performances techniques, les performances (en évaluant de manière subjective la qualité du service rendu), les conseils, les problèmes connus, les niveaux des services, etc.

(b) L'élaboration d'un modèle de Calcul Autonomique permettant au SID d'assurer des fonctions d'auto-organisation : configuration, diagnostic/réparation et optimisation. Ces fonctions sont élaborées autour de l'amélioration des niveaux de service et de la qualité des services. Les boucles du manager autonomique sont décrites en tenant compte des spécificités des entrepôts de données, comme l'organisation hiérarchique ou l'utilisation non-continue.

(c) Le développement de l'approche *BI Self-X*, composée de trois modules, chacun chargé d'une fonction d'organisation. Les résultats obtenus avec cette approche ont montré que les entreprises qui utilisent BI Self-X pour la supervision de leur SID obtiennent des

meilleures performances, ainsi qu'une baisse des coûts et du temps passé dans l'implémentation et la maintenance de leurs entrepôts de données.

Mots clés : Système d'Information Décisionnel, Entrepôt de données, Ontologie, Calcul Autonomique, Qualité de Service

Acknowledgements

This thesis benefited from the insights and direction of several persons.

I would like to thank Mr. Vincent Bénatier for this incredible opportunity of a CIFRE. He was always there supporting me in both the good and the down moments. His contribution to this work and to my professional and personal evolution was immense, and without him none of this would have been possible. I also thank him for giving me the complete freedom of expressing myself and of working even on the craziest ideas, a thing so rare and most valuable nowadays.

I owe my deepest gratitude to Mr. Henri Briand and Mr. Remi Lehn, my PhD supervisors for all their guidance, effort and advice they have provided over these years. They inspired me as a model for research and teaching, as well as for the passion for discovering and learning. Thanks to them, I have gained valuable experience and knowledge, and their role was equally vital in the outcome of this thesis.

I would like to specially thank Ms. Pascale Kuntz, director of the KOD team for all her professional and personal support, offering me the great privilege of being part of the team she pilots. Since the first time we have met, she has been a continuously model for me, and provided very valuable contributions to both this thesis and my scientific development.

I wish to express my gratitude to Mr. Jérôme Darmont and Mr. Jean-Marc Petit for the honor that they made me by accepting the review of this thesis and for their constructive feedback. Equally, my thanks go to Mr. Gilles Venturini for the honor he provided by accepting to be the jury president.

I would like to thank the SP2 Solutions team for supporting my researcher madness, always being there with their support, laughter and joy. These years would have been a lot harder without the support of my colleague and dear friend Claudia Marinica to whom I thank deeply.

In all the ups and downs that crossed my way during the PhD years, I knew I had the support of my friends, be they near or far. I will not name you here, because of the risk of a second manuscript, but its all about you. Nevertheless, I would like to thank one special person – he was always there, listening and encouraging me, and always understanding.

Last, my upper most gratitude goes to my mother, for she is the reason of whom I have became. Te iubesc.

Table of Contents

List of figures

List of tables

Glossary

- AC = Autonomic Computing
- ACM = Autonomic Computing Manager
- AI = Artificial Intelligence
- BI = Business Intelligence
- CDM = Common Diagnostic Model
- CIF = Corporate Information Factory
- CIM = Common Information Model
- CHOP = Configuration, Heal, Optimize and Protect
- COBIT = Control Objectives for Information and related Technology
- COD = Equipe partie de LINA, COnnaissance et Décision
- DBMS = Data Base Management System
- DMTF = Distributed Management Task Force
- DSS = Decision Support System
- DW = Data Warehouse
- GIF = Government Information Factory
- GUI = Graphical User Interface
- IS = Information System
- ITIL = Information Technology Infrastructure Library
- KB = Knowledge Base
- LINA = Laboratoire Informatique de Nantes Atlantique
- MAPE-K = Monitor, Analyze, Plan, Execute over Knowledge
- OLTP = On-line Transactional Processing
- OLAP = On-line Analytical Processing
- OKMS = Ontology-based Knowledge Management System
- OWL = Web Ontology Language
- QoS = Quality of Service
- R&D = Research and Development
- RDF = Resource Description Framework
- ROI = Return over Investment
- SLA = Service Level Agreement
- SLO = Service Level Objective
- SPARQL = RDF Query Language
- SWRL = Semantic Web Rule Language
- TCO = Total Cost of Ownership

1 Introduction

"Even a 2,440 mile journey begins with a single step"

Sun Tzu

1.1 Research context

Information Systems and Decision Support Systems

Information Systems (IS) have developed a lot during the last decades, along with the increase in the quantity and complexity of data. If in the early days of IS foundation, there was not a clear separation of IS by their type and purposes, nowadays, lines are very well drawn, and each IS type offers an entire area of academic and industrial research. The most interesting separation, from our position as Business Intelligence experts, is the one between operational and decision support systems, presented in detail by W. Inmon in his book, [Inmon (2005)]. Operational systems are by their nature designed to store and fast write large amounts of raw data. On the contrary, decisional systems load and transform the operational data into data warehouses, containing aggregated data which serve as a base for the decisional processes. This process of data aggregation and analysis, alongside the process of decision making based on this data, is known as Business Intelligence.

From our background and active working area, we approach in this thesis the issues of managing DSS and data warehouses, as the core of the DSS. The literature reminds two fundamental data warehouse models [Inmon (2010b)], quite different in architecture and purpose: (i) the Kimball model, for a fast deployment, where data warehouses have no apparent architectural model, suitable for small projects and (ii) the Inmon model with a well defined data warehouse organization on several hierarchical levels, suited for more advanced complex projects. Due to the increased complexity of the latter and the fact we have came across it most of the time in our experience, we have chosen it as the data warehouse architecture model with this thesis.

Despite the differences between the two, the issues faced by the decisional world remain valid in both cases. One of the biggest mistakes in enterprises today is the application of operational management strategies on decisional systems without taking into account their specific characteristics. Elements such as different utilization patterns or integration of service level agreements with the management processes are one of the major lacks of current DSS implementations. Moreover, if CMDB management guidelines such as ITIL or COBIT are very detailed for operational systems, they are not so well specified for decisional systems [Dumont (2007)]. The main reason behind this is the increased level of subjectivity when it comes to BI, due to the fact that data warehouse performance should be measured not based on technical indicators but based on the quality of service as a direct indicator of the users' satisfaction. Focusing on improving solely technical performances can be a recipe for disaster. This subjective aspect makes management policies a lot harder

to define and follow, thus requiring constant human intervention for task resolution, leading to high amounts of spent time and money.

In this context, there are two research fields that grab our attention: (i) manual/autonomic task resolution and (ii) knowledge formalization and integration. They are not disjoint, on the contrary, the first rapidly implies the second. Task management refers to the elaboration and delegation of tasks such that they are executed by machines instead of humans. From the lack of proper management policies, tasks with DSSs are almost exclusively performed by humans, even the most basic low-level tasks, such as parameter configurations. Apart form the repetitive aspect, by focusing on these tasks first, DSS experts have less time to allocate to higher level tasks, such as the implementation of business objectives. In order for these low level tasks to be executed by a machine a proper autonomic model should be integrated but with the characteristics of DSS in mind. One such solution we have investigated is Autonomic Computing [IBM (2006)]. At the core of the autonomic computing model are knowledge bases. Combining this with the subjectivity aspect of describing DSS management policies, we have explored the usage of semantic web technologies and ontologies for knowledge formalization and representation.

Managing IS: Autonomic Computing

Introduced by IBM in 2001 [IBM (2001)], Autonomic Computing has the objective of helping IT specialist fight the biggest challenge of IS: complexity. As self-stated, it aims at leveraging experts of the low-level repetitive tasks, by an autonomic model which permits their execution by the machines. Inspired by the functioning of the human body, AC provides a self-management model, by identifying four principles: self–configuration, self-healing, self-optimization and self-protection.

The AC adoption cube [Parshar and Hariri (2007)] provides three axis of development with the adoption of autonomic behaviors: control scope, functionality and service. Control scope describes the level of autonomic control with the system, starting from sub-components up to the entire system. Functionality refers to the level of autonomy, from simple monitoring to intelligent closed loops. Service flows contain the systems capability of offering new services, evolving along with the control scope and the functionality.

The central point of the AC architecture is the autonomic computing manager, the entity that implements the AC principles. The autonomic manager contains an intelligent four phase closed loop: monitor, analyze, plan and execute, based on a knowledge source at its ‘center’. This is also known as the MAPE-K loop. Following the human-body functioning parallel, the monitoring phase permits the observation of a series of symptoms. Then, by passing the symptoms to the analyze phase, the manager builds the diagnostics. Last, by integration of the diagnostic and the potential heals, the planning phase organizes the order in which heals are executed while the execution phase acts with these heals. Additionally, and the beginning and the end of the MAPE-K loop there is a series of sensors (before monitoring to gather the data) and, respectively, effectors (after execute to ensure action execution). Passage between the phases of the autonomic manager is usually assured by rules dependent on the knowledge base, often expressed under the form of ECA rules [Huebscher and McCann (2008)].

The AC adoption model includes a hierarchical organization of the autonomic managers, by separation of (i) the managed resources, (ii) AC purposes, (iii) discipline and (iv) human interaction. The first level contains the managed resources autonomic managers, which aggregate to touch point autonomic managers organized by the four autonomic purposes. Next, climbing the hierarchy, these are regrouped by various disciplines within the system, such as, last, on top of the ladder, we have the manual autonomic manager, which provides the tcontrol over the functioning of the autonomic system via the user interfaces.

Nevertheless, the AC model was elaborated with regards to operational systems. Even if business objectives are reminded, elaborating clear scopes and integrating SLAs for improving the quality of service is one lacking aspect of autonomic computing [Huebscher and McCann (2008)]. Therefore, the adoption of AC with DSSs requires special attention. Relating with the knowledge base formalization and the issues of rule management, one possible solution are semantic technologies, specifically ontologies. The path of using ontologies with autonomic computing has only been slightly explored, [Stojanovic et al.(2004)], and there are no references, to our knowledge, of combining this with DSS and data warehouse management.

Knowledge representation: Ontologies and the Semantic Web

As a research field, knowledge engineering focuses on extraction, representation and usage of knowledge, integrated via knowledge bases. This has its roots with artificial intelligence, as knowledge manipulation implies the process of knowledge sharing, reuse and most important inference (or deduction of new knowledge from existing facts).

Relating with the AC model, it is well known that the efficiency of the AC adoption is directly related with the quality of its knowledge bases. With the expansion of the Semantic Web and the concept of linked data, ontologies have become a popular way to model knowledge with complex information systems. Introduced in 1992 by T. Gruber for IT, an ontology defines a set of representational primitives with which we can model a domain knowledge or discourse' [Liu and Özsu (2008)].

From the seminal work of [Maedche et al.(2003)] introducing ontologies for knowledge base formalization has been intensely studied. One of the very important aspects of ontology usage is the capacity to unify different forms of information: from technical documents to human expert knowledge. This is achieved by relying on very high expressivity and on the reasoning capabilities offered by inference engines. An inference engine (or reasoner) is defined as a software program able of deduce new facts or knowledge from known facts by using inference rules. Literature offers extensive examples of inference engines such as Pellet, Racer or Fact++.

With the development of the World Wide Web and the W3C consortium initiated by Tim Berners-Lee, several standards have emerged for the representation of ontologies. The most common are RDFS and OWL [W3C (2010b)] (with their sub-languages and extensions), both XML-based. Consequently, a multitude of dedicated tools and APIs have developed around these, such as Jena, Protégé or the NeOn Toolkit. Moreover, the industrial

world became aware of the potential and impact of semantic technologies, reinforced by implementations such as the Oracle 10g RDF Module or the TopBraid Composer.

The basic structure of an ontology is the sentence, represented under the form of a triplet (subject, predicate, object), interlinking concepts. The totality of sentences builds the ontology graph, where subjects and objects represent the nodes and the predicates the arcs. In turn, a concept which is a predicate in a sentence may become subject or object in another one and so on. As inter concept relations may present several mathematical characteristics (such as symmetry or transitivity), a distinction is made between asserted and deduced (inferred) graphs. An asserted graph contains only the concepts and relations explicitly defined in the ontology. An inferred graph will contain relations that are deduced based on axioms and inference rules, build with the help of a reasoning engine capable of understanding these axioms.

The flexibility and dynamics of the ontology data model permits a much better knowledge integration, thus suiting the issues of DSSs relating to subjective management policies and SLA integration. Moreover, it provides a solid base for AC knowledge base representation, as no specific directions are given by IBM in the AC blueprint.

1.2 Proposed approach and contributions

In this thesis, we have tried to solve some of the issues presented, by providing (i) a unified data model for formalization and integration of the knowledge implicated in the management of DSS and data warehouses, and, (ii) an autonomic adoption model which makes use of this knowledge in order to assure lower management costs and increased service levels. We have named our solution BI Self-X, indicating a self management business intelligence process.

Building a DSS management knowledge base is not a trivial task. We have chosen the usage of OWL ontologies because of the increased expressivity and inference capabilities, which otherwise are difficultly achievable with traditional data base approaches. The DSS management knowledge base is divided into several sub components.

First, we have proposed a description of the DSS and data warehouse architecture, by identifying three knowledge sources: the architectural organization of the DSS, the configuration parameters and the performance indicators. The DSS architecture is represented as a hierarchy of elements (from low level programs and multidimensional bases, to physical servers and the whole DSS itself). The configuration parameters are straightforward, objective and generic (unless specific DSS implementations). The performance indicators are separated from the configuration parameters because they integrate both objective and subjective aspects. The objective performance indicators are the raw technical indicators, specific to operational systems. Due to the DSS context, the subjective performance is given by the levels of user satisfaction, thus depend on several variables (such as the utilization patterns or SLAs). Using ontologies as a data model brings increased value to the solution, especially by usage of transitivity (for the DSS hierarchical architecture) and of inference rules (for performance and SLA integration). We keep the DSS model as generic as possible (with the occasional implementation specifics).

Second, we focus on integrating all the management information, with regards to the autonomic computing model. This time, ontologies show their real power, as we need the integration of information coming from completely different knowledge sources. Among others, these include technical readme documents, technical forums or human experience. Our proposition of integrating all these under the same unified knowledge base for DSS management is a completely novel approach. We consider that the ITIL management guidelines with the CMDB need to climb on a superior level. Our A-CMDB proposition, prior to the works of this thesis was just the first step. The second step is actually the definition of the CMKB (Configuration Management Knowledge Base), as a logical evolution of the database model with the power of semantic technologies.

The adoption of the AC model with DSS requires special elaboration and implementations, and our proposition towards this is, to our knowledge, the first. Due to the characteristics of DSS and lack of SLA adoption, our AC model proposition uses specific self-management heuristics, based on the CMKB. We have focused our efforts towards the self-configuration, self-healing (including self-diagnostic) and self-optimization principles. Moreover, in addition to IBM's hierarchical model, we have added horizontal links at each level of autonomic computing managers. This was possible by using the capabilities of ontologies, via inter concept links and inference engines, while we equally constructed an ontology model of autonomic computing. The communication, that assures both the individual MAPE-K loops and the inter computing manager links, is described with the help of inference rules on top of the DSS ontology model and the AC ontology model.

The presented results have the objective of showing how the implementation of the AC principles with DSS management, provides lower costs, higher availability, better performance and, most important, increased user satisfaction. The results are presented for each of the three AC principles, for both theoretical and real client environments. The BI Self-X self-configuration module shows how integrating business rules and best practices helps the deployment of new data warehouses, by providing better starting configurations and implicitly increased service levels. The self healing module, especially through its self-diagnostic module, provides a fast and reliable DSS diagnostic, reducing the enterprise financial costs and data warehouse unavailability times, while equally gaining human experts precious time. It is the perfect example of how autonomic computing helps the human with his tasks. Finally, the self-optimization module performs continuous monitoring and analysis of the DSS, while trying to optimize the resource / performance ratios, based on criteria specific to DSSs. The usage of the heuristics is proven here by their implementation in the loops of the autonomic computing managers.

Overall, from the proposed approach and obtained results, the fundamental objective of our proposition is reached: the DSS management costs are lowered while the BI process becomes more efficient and user satisfaction increases. We equally hope that both the ACMDB and CMKB notions are adopted as future references for DSS management, and that the paths we have explored excite and offer future research (us included) perspectives.

1.3 Thesis outline

The thesis is organized into three major parts: (i) the state of the art, (ii) our approach and (iii) the results we have obtained. In detail, it has the following outline:

State of the art

Chapter 2 presents an overview over the management of information systems today. It provides the definitions of the main concepts, the research areas that are concerned with IS, the management models and software and the principal management procedures and references.

Chapter 3 introduces the notion of Autonomic Computing as a potential answer to the complexity of IT management. The definitions and the AC principles and goals, as well as the integration with the existing management practices are first introduced. Then, the Autonomic Computing Manager is shown as the core of the AC adoption as the various types of ACMs are brought into discussion. Finally, there is a discussion over the comparison between AC and Artificial Intelligence systems.

Chapter 4 is dedicated to knowledge management, as a continuity to the requirements of IT management and AC models to implement knowledge bases. It introduces first the notions of web semantics, and how semantics can change the way information is understood today in IS. Then, it provides a detailed description of ontologies, with the main technologies and evolutions to this field as well as its applicative areas. Last, it relates to the existing works of how ontologies and AC work together for better IS management.

Chapter 5 is concerned with a specific type of IS management: decision support systems and data warehouse. The described elements form the applicative environment of this thesis. DSS concepts are explained, following with the DSS information core: the data warehouse. The challenges and problematic points are presented as we try to bring them answers with our approach.

Approach

Chapter 6 describes the current solutions provided by the actors implicated in this thesis. The industrial environment is presented as well as the context from which this thesis was born, with the solutions offered by SP2. Then, a series of issues from the academic and industrial worlds are raised, as we focus on solving them, and culminate with the detailing of two use case scenarios.

Chapter 7 proposes the BI Self-X approach to managing DSSs. It is divided into two major sub sections, one about modeling the DSS environment (from a conceptual UML point of view) and the other about implementing and adopting AC and ontologies to transform the UML models to semantic models. The DSS environment is divided into three main parts: architecture, configuration and performance indicators (static) and best practices (dynamic). Further, it shows how non structured information can be transformed into structured information by detailing a UML-OWL transition. The ontology AC adoption

proposition is last presented, as to understand how our approach combines the two elements with the DSS specifics. A series of rules and two heuristics are detailed to this end.

Experiments and results

Chapter 8 presents some of the most notable experiments and results we have obtained. First, the experimental device and the software prototypes are presented. Then a discussion over the experimentation methodology is done, with a clear distinction between the theoretical (laboratory) and the real (client) environments. Last, following the first three principles of AC, for each of the two environments the obtained results are detailed and interpreted.

Chapter 9 concludes the thesis, by describing a series of conclusion points and the perspectives we have for our future works. Conclusions are presented for each of the aspect that has been approached in the thesis, whereas the perspectives make reference also to other projects that are development and that, we hope, will contribute to the advancement of these new technologies.

2 Information Systems Management

"Information is not knowledge"

A. Einstein

2.1 Introduction

Information has been the constant source of human evolution throughout time, and access to information is considered today, in the modern society, the true source of power [Toffler (1991)]. The development of recent technologies put the human individual in the position where he can manipulate and present information in a form that would create different perceptions to different individuals. Subsequently, there is a direct link between information organization and people organization: we organize information the same way we organize people. Information systems, in computer science, as a part of the information management area, have a long history with Management Science, and are the pioneers in what IT is known to represent today.

This section makes on overview over information systems management, with the notable references from the literature which are interesting aspects for this thesis. First, a state of the art with the definitions and the main concepts of IS is made. Second, management software for IS is presented, with the main categories such as ETL, GUI etc. Then, a review of the IS management procedures is done, with relation to the main norms and guidelines. Last, a set of conclusions is presented, with the overview of current problems and bottlenecks of IS and how some of them may have an answer in some of the elements presented in this thesis.

2.2 Information systems concepts

Information Systems (IS) is a very large term, and thus is hard to define specifically.

Definition: In computer science, information systems are defined as the mediator between the humans and the computer as describing the interaction between the people, the processes, the data and finally the technology [O'Brien (2002)].

IT refers to anything related to the computing and computer technology including: networking, hardware, software, internet and, of course, humans. The adoption of IT is now worldwide spread, as enterprises usually have their own specific IT department; the experts working here are in charge of any tasks related to a computer.

Research in the field of IS debuted in the mid 60s, with the initiative of Harry Stern, who in 1967 started a column with the topic of 'Information Systems in Management Science'. This proved to be a good support for future discussions, leading to the introduction of IS in management science in 1969 as a first departmental structure, divided into: IS Applications and IS Theory [Ackoff (1967)]. Some of the points stated back then are nowadays obsolete, such as:

- The lack of information to management level – today it is the contrary, as there is too much information, and the problem is how to select 'the needle from the haystack'.
- If managers get the information they want then decision will be better – today managers improve their decisions by asking all the time information derived from what they initially thought to suit their needs. It is a continuous cycle.
- Understanding of the system is optional at management level – not entirely true as one cannot critically evaluate the system if he doesn't understand at least its basics.

2.2.1 Information systems research areas

There are several IS problematic topics, that cover a large spectrum of information systems. These topics include: system design, inter-organizational management, IT evaluation, performance measurement, return over investment and creation of business value by IT etc. Therefore, five main research areas have developed: (i) decision support systems and system design, (ii) value of information, (iii) human-computer system design, (iv) IS organization and strategy and (v) the economics of IS and IT. They can be seen in the figure below, as presented by [Banker and Kauffman (2004)]:

Research stream	Level of analysis	Theories	Methodologies used	Related disciplines
Decision support and design science	System level, mostly in conjunction with human users or business processes, up to the level of a strategic business unit	Decision theory, network optimization, control theory	Mathematical programming, forecasting, simulation, expert systems	Computer science, operations research, economics, marketing, strategic management
Value of information	Individual decision makers, technologies in business process context, firm actions in market context	Information economics, real options theory, information sharing theory	Decision trees, analytical models, statistical analysis, mathematical programming, simulation	Economics, decision science, risk management
Human-computer systems design	User focused, involving both individuals and groups	Cognitive style, behavioral decision theory	Experiments, argumentation, simulation, system test-beds	Cognitive psychology, decision science, design science
IS organization and strategy	Spans levels: individuals, groups, business units, organizations, marketplace	Diffusion theory, media richness theory, resource-based view of the firm, transaction cost economics, task-technology fit, technology acceptance model	Models, case studies, field studies, experiments, surveys, cross-sectional and longitudinal designs, argumentation, blend of qualitative and quantitative methods	Organizational theory, strategic management, social psychology, cognitive psychology, economics
Economics of IS and IT	Spans levels: individual decision makers, business process/product/project, strategic business unit/firm, industry, market, economy	Theory of the firm, production economics, game theory, contract and incomplete contracts theory, network externalities	Analytical modeling, empirical analysis and econometrics, cross-sectional and longitudinal design, experiments, simulation	Economics, operations research, computer science, strategic management

Figure 1 : The five streams of IS [Banker and Kauffman (2004)]

Decision support and design science – the area that put the bases of the modern decision support systems and design interfaces. Decision support expresses very well the notion of human – computer interaction, as reports build from the decisional data enable experts and managers to better run their business. The level of analysis is thus presented in conjunction with the human side, as it plays a very important role in the process of decision making. Theories and methodologies of decision support and design include decision and control theories, advanced data mining operations, prediction and forecasts, simulation,

expert systems etc. A good part of decision is based on the evaluation of past trends and the prediction of future behaviors based on the available data.

Value of information is quantified as the difference in payoff, when a decision expert knows certain information and when the same information is not available to him. There are several levels of information, two main types being indicated as reference: *imperfect* and *perfect* information. *Imperfect information* is useless, incomplete or false information; generally it's the information for which the payoff is small. In worse case scenarios the absence of this information is preferable to false or incomplete information. *Perfect information* is the information for which the payoff is maximized, giving maximum value to the decision maker. The analysis level is specific to individual decision makers, with theories revolving around informatics economics and information sharing. The main topic remains the decision science.

Human – Computer Interaction was developed with the aim of helping solve a core problem with IS: "management misinformation […] the user suffers more from an over-abundance of irrelevant information than from a lack of relevant information" [Ackoff (1967)]. By linking to studies in psychology, identifying individual differences and cognitive types, development of human computer interfaces is studied to help humans work with IS. Therefore, the level of analysis is user or group focused, as the human is the sole benefit part from an HC interaction interface. The theories revolve around the cognitive style and human behavior, with methodologies from experiments and simulations about human reactions.

IS Organization and Strategy emerged as IS researchers realized that there are several levels of analysis: system level, business process, strategic business unit and organizational level. Thus, it covers all analysis aspects from single individual to organizations and marketplaces. Theories include transaction-cost economics, task-technology or technology-acceptance models, case studies or surveys. Organization and strategy involves both the social psychology and the organizational and strategic management theories.

Economics of IS and IT focuses less on the strictly technological aspect and more on the firm theory and IT economics value. This stream brings together the organization, the markets and the industries, enabling IT-coordination between the 3 major 'players'. Regarding the productivity impacts and business value of IT, a paradox was noticed, as effects where not always positive [Brynjolfsson and Hitt (1996)]. Yet, the way the ROI and payoff were calculated for IT didn't include aspects such as product quality improvement or product variety, which are also influenced by IS and IT departments.

2.2.2 Corporate Information Factory - CIF

One important part of IS, from the decision area, is the *Corporate Information Factory (CIF)*.

Definition: CIF is defined as a logical architecture with the objective of delivering business intelligence and business management capabilities driven by data provided from the business operations [Imhoff (1999)].

CIF is business oriented, to provide a strategic and tactical support for decision systems, and was first introduced by W.H. Inmon. Recently a derivation from the CIF was described by the same author: the *Government Information Factory (GIF)* [Inmon (2010c)] which related to government IS architectures. Its principal purpose is security (as it was developed soon after the events of 9/11), and the main difference with the CIF are the actors involved with building the GIF data warehouses. CIF data concerns the single (or consortium) enterprise system, whereas GIF data is intended to be accessible to all governments in all countries (a global data architecture). Other differences include the need of accommodation for very long periods of time with GIF and the stricter security levels.

The elements of the CIF architecture are presented in Figure 2, as adopted from Inmon's model:

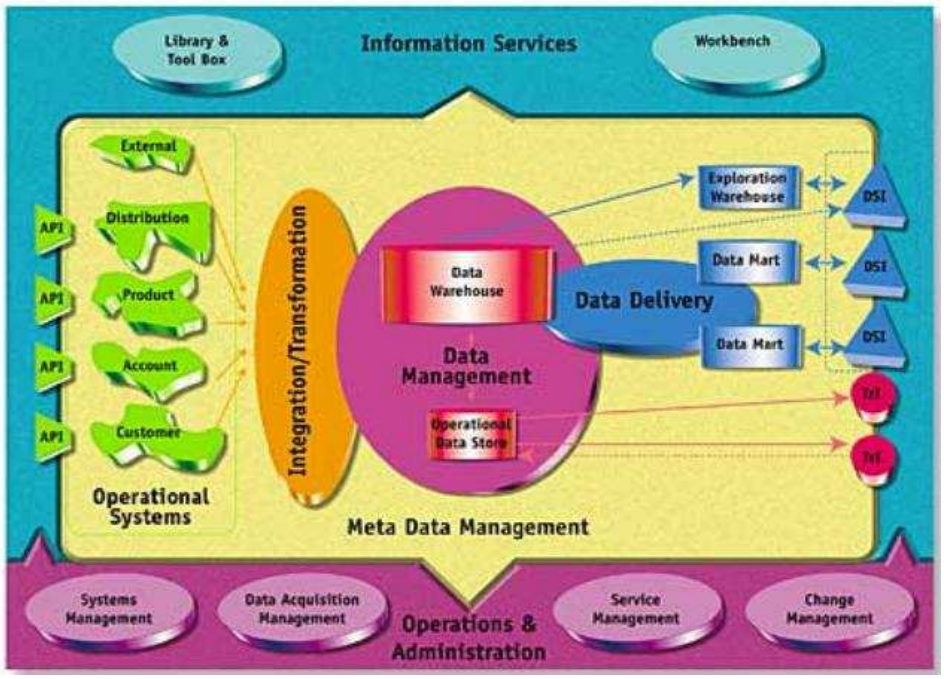


Figure 2 : The Corporate Information Factory [Imhoff (1999)] adapted from [Inmon (2010a)]

There are three main parts: (i) *Information Services*, (ii) *Meta Data Management* and (iii) *Operations & Administration*. The information flow passes from the Operational Systems via integration and transformation into data warehouses and data stores. Basically, there is a formatting and standardization of the operational information such that decision data management is assured in an effective way. Once the data warehouses are build, the data delivery will provide with data for the DSI, and for the interfaces for data analysis. Then an entire chain of Operations & Administration processes is done over the available data. Some of the notions and the elements that are presented in the CIF diagram will be more detailed in future sections, in particular the decision support aspect (data warehouse, operational data store, data mart etc.)

- *Operational Systems* – are the core systems that support daily business operations. APIs facilitate the access, and data from operational systems is the raw primitive

data used later for the construction of the data warehouses. CIF takes its characteristics from the operational systems, as they were the first described IS. The data that is included here comes from several different sources, and it includes both formal and structured data (e.g. databases, spread sheets) and informal and non-structured data (notes, emails etc.).

- *Integration and Transformation* – includes several steps, before the data is 'ready' for the data warehouse. In this process the data is usually captured, filtered, transformed into a single unified form, reengineered and afterwards loaded into the data warehouse (or the operational data store). The process is critical, as it transforms 'chaos into order'. Operational data is not organized and is distributed, while analytical data is well architectured and stored into specific data warehouses.
- *Data management* regroups the two ways of storing analytical data
    - *Data warehouse* – is the 'place' where all the transformed operational data will be stored. It is subject-oriented, integrated, temporal, non–volatile and provides the data used for decision support systems or business intelligence (*[Imhoff (1999)]*)
    - *Operational Data Store* - is the operational data warehouse, using the same definition with the observation that ODS is *current* instead of *temporal*. While a data warehouse contains evolution over time data, an ODS will contain data from a specific point in time.
- *Data delivery* – is the first interface that allows users and decisional experts to obtain views with the data from the data warehouse or data marts. A data mart is a smaller collection of data from the data warehouse, with the purpose of assuring a specific business function. Data delivery basically describes the process of building a data mart, which consist of 3 steps: (i) *filter* , (ii) *format* and (iii) *delivery*
    - *Filter* – removes all the unneeded information for the data mart.
    - *Format* – puts the filtered data from the data warehouse into the specific schema of the data mart (e.g. star schema, cornflake schema etc).
    - *Deliver* – once the data is in the data mart format, it ensures that the information is properly delivered to the business users.
- *Decision Support Interface (DSI)* – provides the user or the decision expert the tools to efficiently view and use the data from the data warehouses and the data marts.

In conclusion, the CIF consist of two main 'actors': the *producers* and the *consumers*. The *producers* are in charge of getting the data in (into a specific environment, into the enterprise etc.). Anything that is alimented by data in the CIF comes from the producers. The *consumers* use the data provided by the producers and get it out by assuring decision support and business intelligence.

2.3 IT management models and software

As IS developed, an abundance of research and industry standards, tools and models flourished, such as UML, ETL or MDA. The most important are described as follows to provide a better understanding over IS management.

2.3.1 Extract Transform Load

Extract Transform Load (ETL) is a common terminology which stands for extracting data from source systems, transforming it according to various guidelines and loading the transformed data into other data storages [Centre (2010)]. This process is usually adopted by decision support systems, with data warehouses loading. Yet the principle of ETL is a general principle for all IS, and any software product that provides the three functionalities can be classified as an ETL.

An ETL is used when faced with big amounts of data (i.e. massive data bases) and massive transaction streams. A real life cycle of an ETL, as shown by [Selectorweb (2010)], contains no less than 10 steps: cycle initiation, build reference data, extract from sources, validate, transform, stage, audit reports, publish, archive and cleanup.

The performance of ETL must be very high, (~ several TBs/ hour), while the bottleneck with these software is usually the data loading operation. From the point of view of processing, there are three main types of parallelisms with ETL applications:

- Splitting a big file / table into several smaller pieces and treating them in parallel
- Push data through parallel operations and components that work in parallel
- Run multiple data streams in parallel

Most of the 'big fish' from the IT software industry have developed their ETL software, among which we find: Oracle Warehouse Builder (OWB) by Oracle, Data Integrator & Data Services by SAP, IBM Information Server (Datastage) by IBM or SQL Server Integration Services by Microsoft [ETLtool.com (2010)].

2.3.2 Unified Modeling Language

Unified Modeling Language (UML) is a system modeling tool that offers a general world wide accepted standard for describing a system or software application. UML was released in 1997 by the Object Modeling Group (OMG) with the purpose of providing the (development) community a stable and common design method that could be used to develop and build computer applications [Bell (2003)]. The authors that introduced it were Jim Rumbaugh, Ivar Jacobson, and Grady Booch, which brought together each of their individual view over system modeling into a common effort.

UML offers several advantages, the most important being that it is platform or language independent. This is why it is considered to be more of a meta –language, rather than a methodology. It doesn't require any formal artifacts, and can easily be integrated into a system without the need for change of the existing elements. There are several UML concepts and diagram types that are most often used by both individual users and enterprises, as shown by [Pender (2003)].

Use Case Diagram – a use case diagram shows a function or a unit provided by the system. It describes with the help of actions and reactions the behavior of a system from the point of view of the user (which can have different roles: end-user, system expert, DBA

administrator etc.) [Belleil (2009)]. An UCD permits to define the system limits depending on its uses and users and also to draw the frontier between the inner system and the exterior world.

An UCD basically contains four concepts: (i) the *actors*, (ii) the *use cases*, (iii) the *'include'* and (iv) the *'extend'* links between the use cases. The actors are the entities that interact in a use case scenario, either external or internal. They can be human actors (e.g. a GUI user in front of a web page) or non-human (e.g. a web service provider). The use cases represent the actions that the users can make, directly or indirectly. Use cases are linked with the help of two relations. The 'include' relation expresses the fact that a source use case has also the behavior of a destination use case (e.g. 'Validation of a payment' includes 'Paying'). The 'extend' relation indicates that a source use case develops more behavior over a destination case (e.g. 'Having a reduction' extends 'Ordering a product').

Class diagram – a class diagram describes the entities presented in the use case diagrams and the relations that exist between these entities in a detailed level. It basically shows the static structure of the system. A class diagram contains three concepts: (i) the *class / sub class*, (ii) the *attributes* and (iii) the *operations*.

A class is a representation of an entity, and in the UML specification can have several stereotypes such as: enumeration (a list of predefined elements), entity/Jacobson (a general system class – e.g. the GUI) and control (usually indicating behaviors). There is a class / sub – class relation, which indicates the taxonomy of the system. The attributes are introduced as properties of classes (e.g. a Human has an age attribute), and provide information about the class. Usually the list of attributes permit the identification of the class (having two different classes with the same attributes and operations would not make sense unless certain specific cases). The operations represent the functionalities that the class provides based on its attributes (e.g. an operation of getBirthDate() determines the birth date of a Human based on its age and current year).

Component Diagram – the component diagram provides the physical and logical view of the system's components. For example a component can be a physical server (e.g. Intel Pentium machine with 2 HDDs of 500 GB and 16GB of RAM). In the same time a component can be a software, for example a web server that is installed on a machine, or a web service provider. Component diagrams are used to describe the system at very high and/or very low granularity levels.

Other types of diagram include*: the sequence diagram* used to described the detailed dynamics of a specific use case*; the statechart diagram* which models the different states in which a class can be, and the transitions between these states; *the activity diagram* describing the control flows between two or more classes and usually modeling higher-level business processes; the *deployment diagram* which shows how the component diagrams are physically deployed on various systems.

2.3.3 Model Driven Architecture

With the introduction of UML, new opportunities of system modeling had arisen. Two years after UML was introduced, the same Object Modeling Group presented the *Model Driven Architecture (MDA)* [Group (2010)]. MDA provides a set of guidelines and practices for the structuring of software systems specifications, which in turn are expressed as architectural and construction models. The main goal of the MDA is to clearly separate the problem definition, solutions and software implementations. This assures platform independence and improves the ability of managing a system throughout changes and updates.

MDA is seen as an enhancement over UML as it derives from it, the most significant change being the introduction of semantics with the diagrams. With semantics described, an MDA model offers (i) the capability of checking the consistency and correctness of a diagram and (ii) the ability of generating code automatically from the diagrams.

MDA offers two main model categories: the PIM and the PSM [Bézivin et al.(2003)]. The *Platform Independent Model (PIM)* describes the problem completely independent of the target platform (programming language, middleware etc.). The *Platform Specific Model (PSM)* is specification of the PIM that suits a specific platform, via a series of given transformations. The diagram below shows the transformation from the non formal description of the system to the code model and the target programming language.

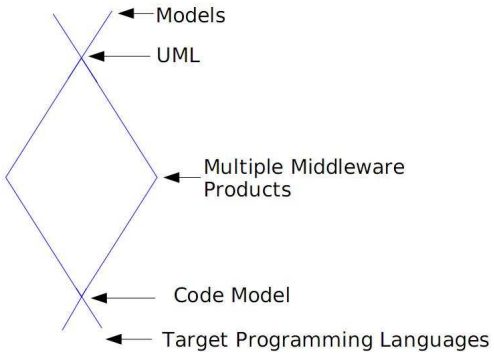


Figure 3 : The MDA transformation diagram [Technology (2004)]

MDA consist of three phases:

- The system is non-formally described which means an ‘on paper’ description, therefore a non-structured format.
- A first formal description (e.g. using UML) of the system is done resulting in a structured format
- From the UML diagrams and the support of the meta model semantics, code is generated automatically, implementing parts of the system semi-automatically (smaller intervention from the developer).

From the perspective of the software developer, MDA can be simply put as: start programming using directly the diagram model and use as little code as possible explicitly.

Examples of MDA tools include the EMF (Eclipse Modeling Frameowrk), OpenMDX (platform for Java J2EE, J2SE and .NET) or Rhapsody developed by IBM.

2.3.4 Common Information Model

The *Common Information Model (CIM)* is an UML based specification for managing information systems. Elaborated by the Distributed Management Task Force, CIM has two components: a Specification and a Schema [DMTF (2010)].The Schema describes the actual model, while the Specification the integration details. The CIM Schema allows semantically rich and object-model descriptions of the managed elements. The CIM model is shown in the figure below alongside its integration with other DMTF solutions.

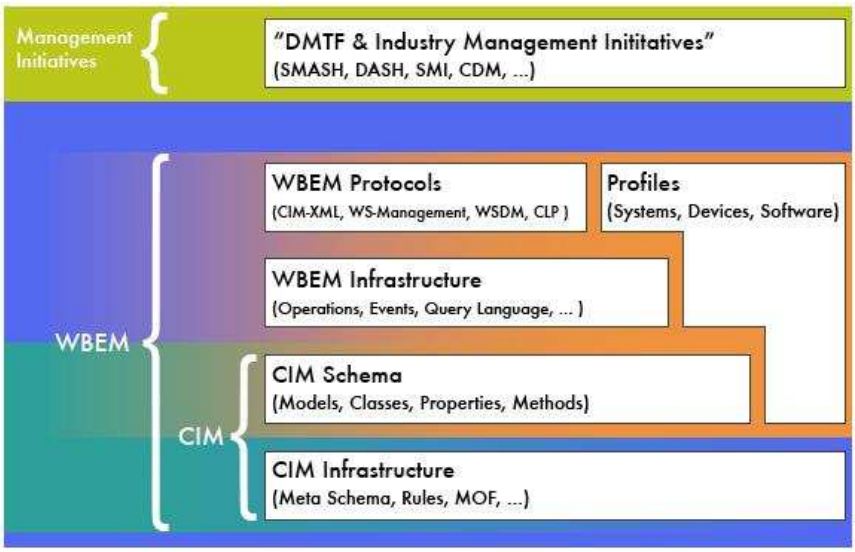


Figure 4 : DMTF solutions with CIM integration [Oasis (2008)]

We note the presence of the CIM at the bottom of the WBEM (Web Based Enterprise Management) model. CIM Infrastructure offers the meta schema that is the rules specifying integration.

On the management initiatives upper level, we find several derivations of the CIM model such as the CDM (Common Diagnostics Model), which is largely used for 'health' evaluation over IT systems. The CDM creates a series of diagnostics, based on the symptoms from the system, diagnostics that are in turn used by other applications on the platform and allow actions such as error resolution or failure prevention.

2.4 Management Procedures

Management procedures for IS have the role of giving guidelines and advice over how to build and maintain the system. These are not obligations but rather directions, from where the terminology of adoption of a management norm (e.g. ITIL adoption [ComputerAssociates (2005)]).

IS management procedures are part of a 'bigger picture', that is 'IT Governance', or how to manage and use IT. This implies specifying the decision rights and accountability framework to encourage desirable behavior in the use of IT [Weill and Ross (2004)]. IT Governance is also defined as the organizational capacity exercised by the board, executive management and IT management to control the elaboration and implementations of IT strategies, in a way to permit the mix between IT and business [Grembergen (2003)].

Two common norms for IT management are: the *Information Technology Infrastructure Library (ITIL)* and the *Control Objectives for Information and Related Technology (COBIT)*.

2.4.1 Information Technology Infrastructure Library

ITIL dates from the late 80s (1989) and derived from a UK Government set of recommendations for IS. Today it contains a series of documents that describe an integrated, process-based, best practice framework for applying IT Service Management (ITSM) to improve operational effectives, delivery of IT resources and the solutions to business needs and requirements [ComputerAssociates (2005)].

The ITIL framework contains three primary segments in term of processes: Service Delivery, Service Support and Other Processes. From these, the first two are the center of the IT Service Management, as they describe the key processes that an enterprise should implement to improve the quality of the services provided. The image below shows the three processes treated by ITIL:

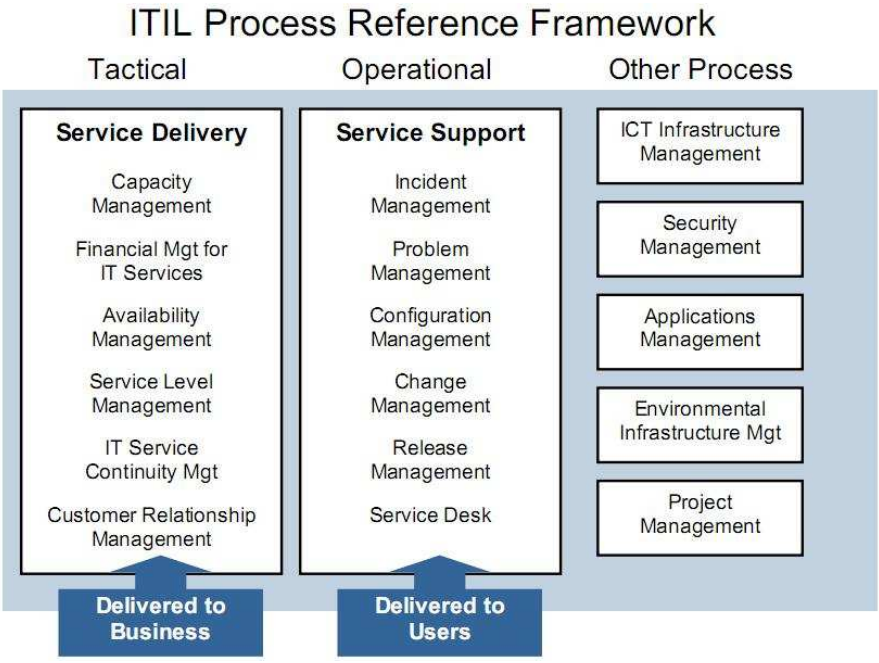


Figure 5 : The three ITIL processes [ComputerAssociates (2005)]

The tactical processes refer to the business delivery, from where the name 'tactical', as it includes capacity management, financial management, ITSM, Customer relationship,

service level management etc. The operational process are client oriented and refer to service support, problem management, configuration management, service desk etc. All of the other processes, such as Security Management or Applications Management, are part of the third category and are not presented as critical.

[ComputerAssociates (2005)] performed a study of ITIL integration and utilization, which was focused on: current practices in ITIL service management implementation; maturity level of the ITIL integrations; drawbacks for a successful ITIL adoption and the usage of ITIL in large enterprises. Their most notable conclusions were, back in 2005

- Only 19% of the enterprises had a mature level of ITIL implementation. The other 81% showed partial or no adoption at all.
- 75% of the organizations prefer suite-based approaches to ITIL. A suite-based approach relies on an integrated suite solution, most having more than they actually needed.
- Big steps over small steps. Enterprises try to directly approach service support and delivery directly without passing through several maturity stages. 50 to 60% of the initiated processes are not implemented.
- The human factor from key management positions are in many situations barriers to an ITIL implementation, because of several causes, such as the lack of awareness (61%) or lack of committed process owners (59%). Therefore, the technical difficulties come in second place as importance with decision making.

2.4.2 Control Objectives for Information and Related Technology

Created few years later than the ITIL, COBIT is an open standard containing a set of guidelines and best practices for IT management. It is aimed at providing alignment between the use of technology and business goals. Organizational goals differentiate COBIT from other existing norms, and its evolution led today at the development of COBIT 5, the latest version of COBIT (in design exposure draft by ISACA [ISACA (2010)]).

There are three levels of implementations, from theory to practice with both academic and business orientation.

- *Level 1* - focuses over IT governance, and how COBIT expresses this notion through control objectives, audit, analysis etc. This is not specific and keeps the generics of the COBIT implementations to a meta-level.
- *Level 2* – is the first level of practical implementation towards a specified environment. Specific IT governance control methodologies are analyzed here, and the choice between COBIT and other norms is put into question
- *Level 3* – represents the actual COBIT implementation, once it is chosen as a control methodology for the solution. Factors such as the industry sector, the size of the organization or the levels of utilization have a direct impact of the implementation.

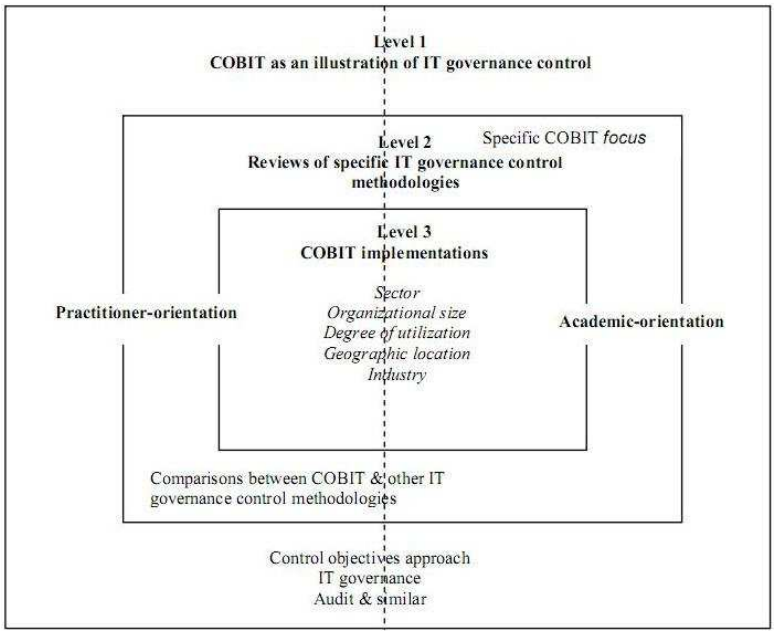


Figure 6 : Framework of COBIT and its Implementations [Ridley et al.(2004)]

2.5 Conclusion

This chapter showed an overview of Information Systems and over the models and procedures of IS management. The principal concepts of IS were presented along with the main research areas from this field. Several models and tools used for IS management were described, such as the UML description language or the MDA or the CIM model. Having the tools allowed detailing the management procedures and best practices for IS, with guidelines from standards such as ITIL or COBIT.

These elements allowed reinforcing the main key issue with modern information systems: complexity. Systems become so big and complex that their management requires more and more human and financial resources. Strategies are required for: improved management, task automatization, autonomic management etc. Many of these strategies revolve around the idea of giving as many tasks as possible to the machines, thus shifting the human workload to computer workload. One such possible solution may be Autonomic Computing, which it will be discussed in the following chapter.

3 Autonomic Computing

"[…] complexity… Dealing with it is the most important challenge facing the IT industry"

P. Horn

3.1 Introduction

Complexity has become the biggest challenge of technology, as the human effort required to manage information systems is increasing exponentially. In the early 1900, when telephony started to develop, human operators were required to switch manually the calls. As it progressed, it was obvious that humans could no longer handle the task (due to the enormous number of switches needed). Therefore, new branches that made such tasks autonomous were introduced to eliminate some of the human intervention needs. The same situation happens today with informational systems.[Huebscher and McCann (2008)] Moore's law concerning the evolution of computing performance is challenged by itself, as the complexity and quantity of information become harder to efficiently manage. Paul Horn resumed in 2001 that: "The information technology loves to prove the impossible possible. We obliterate barriers and set records with astonishing regularity. But now we face a problem springing from the very core of our success - and too few of us are focused on solving it. More than any other IT problem, this one – if it remains unsolved – will actually prevent us from moving to the next era of computing. The obstacle is complexity … Dealing with it is the single most important challenge facing the IT industry"[Horn (2001)]. For instance, it is a usual situation to be faced with tens of millions of lines of code, which require significant numbers of experts and developers in order to install configure and manage the application. In the near future, human expertise alone will not be able to cope even with the every day simple tasks or problems. Therefore, complexity requires more and higher trained experts, to the detriment of costs (and human resources are the most expensive) and the acceptance of fault due to human prone error.

From this perspective, enterprises need an increasing number of IT experts to handle their systems, while these experts are obliged to spend more and more time for simple, repetitive and low level tasks such as maintenance, resource allocation, configuration, error recovery etc. The quality objective changes to assuring at least a maintenance level of the system. Higher level tasks such as service improvement, optimization or quality of service assurance become secondary, although they are the vital point of IS. For example, rather than focusing on how to optimize a data base parameters for data delivery for a customer service application, an IT expert should instead focus on how to improve the application such that it provides the customers with increased functionality and greater conveniences. A study by [Klein (2005)] shows that, in average, enterprises spend up to 80% of their budges on maintaining existing applications and infrastructure. IT experts spend most of their time locating, isolating and repairing problems. Another study by IBM shows that only 13% of the CEOs believed that their organizations is seen as "very responsive" to change [IBM (2004)]. This is alarming, as it stays in the face of the evolution process. We have reached so far just as to try and 'fight' to remain where we are, with no clear horizon of going

further. As Alfred North Whitehead stated: “Civilization advances by extending the number of important operations which we can perform without thinking about them”, and this translates into evolution equals our capacity of dealing with complex systems in a ‘reflex’ way.

Trying to answer the complexity problem, in 2001, Paul Horn, at the time senior vice president of research at IBM, posed the foundations of “Autonomic Computing”. Autonomic Computing is defined as the capability of an IT infrastructure to self-manage itself, with specific goals and policies. It represents a collection and an integration of technologies that lead to an IT computing infrastructure according to the IBMs agenda for the next era of computing – e-business on demand.[Ganek and Corbi (2003)] As specified by the IBM whitepaper [IBM (2006)], in an on demand business IT experts must focus their efforts on service delivery and improving the quality of service, while reducing the cost of ownership (TCO) of their operating environments. Nevertheless, Autonomic Computing does not seek to eliminate the humans from the equation, as some may misinterpret. It simply tries to formulate a new infrastructure in which both human and machine (autonomic process) work together. Its goal is to help the human expert with his daily tasks and workloads.

3.2 Origins and evolution

The source of inspiration for the Autonomic Computing architecture is, as a paradox, the human itself. The human body is a very complex system in which many of the tasks are done in the form of non-aware actions. Reflexes such as breathing or heart pulsation are part of our daily life and the health of the entire body depends on them. Though, we never really think about them or voluntarily control them. The name of the controlling entity, autonomous nervous systems, speaks for itself. The human body is the perfect example where autonomy and manual are combined. We breath all the time without thinking of it (autonomic), when we are hungry we move our mouth so we can chew (manual intervention) and then the food gets digested (again autonomic). Starting from this, researchers and IT experts looked for a way of applying the biology considerations to IS. The objective is to provide IS with self-management capabilities, such as self-healing, self-configuration, self-optimization or self-protection (the self-X factor). The first works were done though in the area of self-healing as it is the vital part of a system. In order to optimize and improve, we must first be able to sustain a healthy configuration. Relating to this, there were several projects that had an influence in the field of autonomic research, as presented from the survey of [Huebscher and McCann (2008)].

In 1997, DARPA initiated a military project called Situational Awareness Systems (SAS)[DARPA (2009)]. The purpose was to create personal communication devices, between the soldiers on a battlefield, so that information about enemy positions, equipment, tactics etc. were up to date between all the combatants. This way a knowledge base about battle reports would be created and synchronized between the participants involved. Data was collected not only from the human part (soldiers) but from environment sensors, automated pilot vehicles etc. Then, the communication of the updated data should not be interfered by the enemy, and should allow communication between all actors involved. For

guaranteeing minimum enemy interception, they used multihop ad-hoc routing, a device that sends data to a neighborly area only, and then in turn, the receiving devices acted as other routing points thus creating chains of transmission until all the receivers have the data. This problem of decentralized routing self-management was a real challenge (with lag limits of 200 ms from the time a message is send to the time the message is received). As for scaling, it was considered for situations up to 10.000 devices on the battlefield.

Another DARPA project relating to self-management was the DASADA. Its objective was to research and develop technology that would allow critical systems to meet high levels of service quality, such as: high assurance, dependability and adaptability. This would be the first pylon of the architecture-driven approach to self-management, and includes sensors and probes for monitoring and system adoption.

In 2001, IBM suggested the term of Autonomic Computing, their whitepaper making the correspondences with the functioning of the human body. IBM stated that IS should, besides the human-side control, implement autonomic management capabilities. They should be able to run independently regular maintenance, configuration and optimization tasks, with the objective of facilitating the work of IT experts (especially system administrators). Four principles relating to self-management are defined: self-configuration, self-improvement, self-healing and self-protection.

Then, in 2004, DARPA started a program called Self-Regenerative Systems (SRS), with the objective of developing "technology for building military computing systems that provide critical functionality at all times, in spite of damage caused by unintentional errors or attacks".[Badger and Todd Hughes (2004)] Four key characteristics relate to the project:

- The software becomes resistant to attacks and errors by generating large number of versions with similar behavior but significant different implementation. This way, attacks are prone to affect only small parts of the entire program.
- Random or algorithmic planned binary modifications are done to the actual code of the software, thus security holes are harder to spot and exploit. There is also a charge reallocation module that shifts away from damage-prone causing resources, that is updated with each resource usage based on the state of the used resource (if a resource is infected it shifts away while trying to recover).
- An intrusion-tolerant replication architecture, that acts as a backup for malicious actions form the inside (for instance an intentioned bad modification, with the correct authorization rights).
- A predictive module responsible for the detection of possible 'inside hijackers' based on their behavior, and capable of blocking them from taking critical damage actions.

In 2005 NASA put the bases of the Autonomous Nano Technologies Swarm (ANTS) project. Their aim was to launch in an asteroid belt 1000 small automated spacecrafts, from a stationary ship, for exploring the asteroid belt. As they expect to lose around 70% of the spacecrafts, an important aspect was the ability of the spacecrafts to communicate and exchange messages. They from small zonal groups, with one leader that gathers the data from its workers and decides what to do further (explore or change area). With several

groups like this, message exchanging is done between the coordinated groups, thus creating a hierarchical tree architecture (as small groups would be in turn clustered to a greater group up to a 'mother ship' or ground control). NASA has used autonomic behaviors in the Mars Pathfinder [Muscettola et al.(1998)], and shows great interest in the autonomic research field as most of the outer space research is done by autonomic probes. One primary cause of this is the long communication lag between earth control centers and outer crafts, thus they need to be capable of self-managing between the commands issued by control centers.

Table 1:- Autonomic purpose projects

Project Name	Project Initiator	Year	Description
SAS Situational Awareness System	DARPA	1997	Decentralized self-adaptive ad-hoc wireless mobile nodes that adapt routing to changing topology of nodes and adapt frequency and bandwidth to environmental and node topology conditions.
DASADA Dynamic Assembly for Systems Adaptability, Dependability and Assurance	DARPA	2000	Introduction of probes in software systems for monitoring, and adapting the system according to the monitored data.
AC Autonomic Computing	IBM	2001	Introduction of self-management techniques based on the functioning of the human body. Based on four central self-x properties: self-configuring, self-optimization, self-healing and self-protection.
SRS Self-Regenerative Systems	DARPA	2004	Military self-healing systems that recover from errors and hijack attacks.
ANTS Autonomous Nano Technology Swarm	NASA	2005	Groups of autonomous spacecrafts, organized in local clusters hierarchies for exploring asteroid belts. The architecture takes into consideration the loosing of 70% of the probes, and must assure functionality within the remaining 30%.

3.3 Principles and goals

The main goal behinds IBMs Autonomic Computing initiative means more than just relating to a single solution provider. The effort should be collaborative such that everyone can benefit from it, leading to an industry-standard management architecture with unified concepts and resources, that works and interacts based on business and service policies. Autonomic Computing proposes an architecture that leads to the development of intelligent, open, self-managing systems. As specified by IBM a self-managing system has three main characteristics. It has to:

- Incorporate knowledge through knowledge bases, about its state, elements, context, activity of the resources from the whole infrastructure etc.
- Be able to observe any changes in the environment, by proactively monitoring individual components and services, searching ways of improving its functioning, noticing change and the impact of change within the system.

- Have the ability of planning changes by modifying its state, modifying other components of the infrastructure or taking actions according to business policies.

No matter how we approach self-management, there is always a final judge of the efficiency and effectiveness of an IT process, and that is customer value. It can be used by measuring the time needed to execute a process, percentage of correctly executed processes or, of course, the cost to execute the process. [IBM (2006)] Thus self-managing systems can and must improve these metrics, by helping improve responsiveness and quality of service, reduce TCO, and enhance time to value through two main aspects: Selective Process Automation and Reduced Time and Skill requirements.

Selective Process Automation – usually these processes require multiple IT experts to manage by: initiating the process, collecting of the data, over viewing the problem records, creating change requests etc. Most of the time the data is not centralized, is incoherent, found in multiple different sources (technical forums, poorly documented documents). Self-managing capabilities help integrate all this data so that the experts can access it in an organized and clear form, helping them with making decisions and better understanding the management processes. Also, components are able to perform certain tasks independently from human intervention, tasks based on the information within the system itself. This leads to the reduction of the number of manual tasks with the benefits of a more precise execution of the processes, more accurate data and a relief for the experts.

Reduced Time and Skill Requirements – the problem so far with complexity is that it needs higher qualified IT personnel, over longer lasting and more difficult tasks. One such task is the 'assets change impact'. This analyses how changes in a system (e.g. OS changing, installation of new services etc) impact the current functioning of the system (i.e. compatibility issues). The 'diagnose problem' is part of the capability of change impact description. In a self-managing system, the knowledge within the system contains the information about the expertise necessary to perform the change activities. This helps reduce the complexity of tasks and the time spent to execute them, while also reducing the number of required experts. In addition, it allows the IT professionals to focus on higher value tasks, such as implementing business policies and improving the quality of service the system provides.

The characteristics mentioned above translate into the four principles of Autonomic Computing, each corresponding to a fundamental area. These areas have the role of brining autonomy and efficiency, by improving system management capabilities, where manual processes and tasks are neither efficient nor effective. The self-managing capabilities accomplish their functions by taking appropriate actions based on what systems sense on their environment. They are inspired by the properties of software agents, identified by [Wooldridge and Jennings (1995)] which are:

- *Autonomy* – the capability of agents to work without external intervention, thus presenting a self control over their actions and states
- *Social ability* – the capability to interact with other agents, of the same or different kinds, by using an agent-communication language (ACL)

- *Reactivity* –the capability of perceiving the environment of the agent and responding accordingly to changes within this environment
- *Pro-activity* – the capability of agents to take actions based on goal directed behaviors.

The four fundamental principles of Autonomic Computing are also known as the *C.H.O.P.* principles (Configure, Heal, Optimize and Protect).

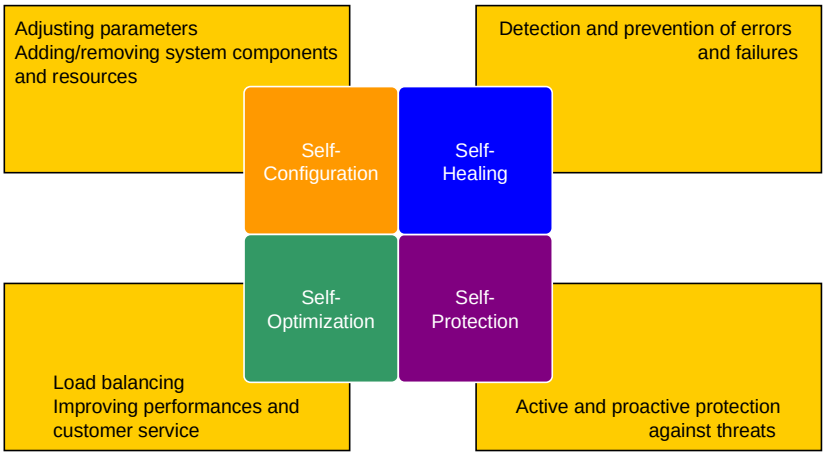


Figure 7 : The C.H.O.P. principles adapted from [Miller (2008)]

Self-Configuration – enables a system to change its configuration and adapt to changing conditions. Examples of such conditions can be: adding or removing components, modifying resources, installing software, integrating new functionalities etc. A system implements self-configuration if it manages to define itself 'on-the-fly'. Similar to plug-and-play devices systems must be designed to enable the capabilities of this feature. In turn, this will reduce human intervention, both on individual components and over the system as a whole.

Self-Healing – represents the capability of a system to prevent and recover from failures or errors. There are two dimensions here, one in the area of prevention and the other on the area of recovery. By prevention, the system can alert about upcoming problems, usually the human side being the final responsible that takes the decision of how the system should change. Recovery means that the system has already been damaged and it describes the strategies that can be taken for healing.

Self-Optimization – enables a system to continuously improve and tune itself. It enables pro-activity over existing elements, thus enabling their improvement, as well as reactivity to environmental changes, thus enabling adaption over given situations. Self-optimization requires software and hardware to efficiently allocate and use resources with minimum human intervention. IBM applies this with local partitioning, dynamic workload management and dynamic server clustering. This is extended to a 'meta' level such that it optimizes all the resource of the system, thus creating a sort of 'logical' workload manager. Similarly, disk storages, databases, networks, CPU, RAM memory etc. should be continually tuned to enable efficiency

Self-Protection – means that a system is able to detect, identify and heal potential threats to its functioning and security. By difference to the proactive aspect of self-healing, self protection deals normally with exterior factors (or internal 'hijacks'), such as: viruses, unauthorized access, network intrusion, denial-of-service attacks etc. Today, such capabilities are even extended to prevent physical protection, an example being the free-fall protection that laptop hard disks can have, by temporarily covering their heads when they sense a sudden change in motion.

3.4 Service management

The four principles are presented over the entire system as a whole. A real system consists of multiple sub components, with different specifications, organization, vendors and communication ways. So, an approach of autonomic computing introduction over each component is laborious and not feasible. Therefore, an approach oriented towards the evolution of service values with the autonomy levels is shown to be more suitable. [Parshar and Hariri (2007)]. It illustrates an adaption under the form of a control scope/functionality/service flows cube

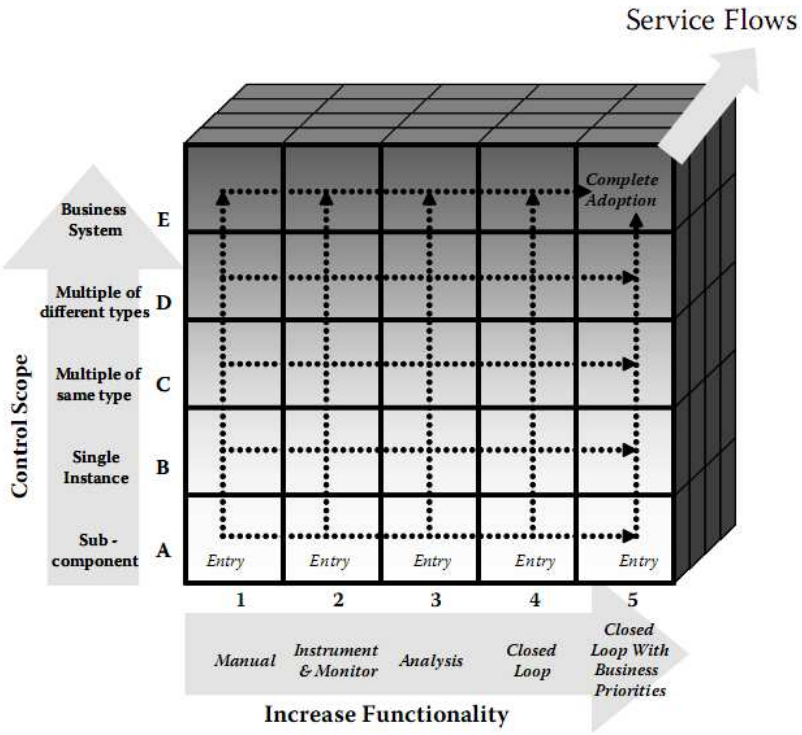


Figure 8 : Autonomic Computing adoption model [Parshar and Hariri (2007)]

There are three dimensions of the autonomic computing adoption cube:

- *Control Scope* – Along the y axis, describes the increase in the scope of autonomic control, starting from sub-components up to the entire business system. The system is organized into a hierarchical architecture (technical component, instance of several sub-components, then several instances, different types and finally the overall business system, as the highest entity of control).

- *Functionality* –Along the x axis, describes the level of autonomy one can obtain. Starts from manual, where no autonomy is present. The next level uses little autonomy with the introduction of monitoring. Then a higher level is achieved by analyzing the monitored events. The fourth level is a closed loop of self-managing but based on the system only, leaving the highest autonomic functionality to the last level where business policies and priorities are introduced in the closed loop.
- *Service flows* – Along the z axis, the service flows describe the systems capability of offering new services. Service flows evolve along with the control scope and the functionality. Coordinating and integrating functions with processes (such as change, performance, monitor, errors, security management etc.), allows the system to cover a larger area of services across all these (different) processes. The objective is to offer the highest number of services possible, while assuring the minimum specified service requirements. To exemplify, creating a group of data bases for an enterprise, to offer a new functionality of archiving its data, requires service from several processes: installation of the data bases, management of security with the data base access, controlling the version mechanisms for backups and recovery and change management at service level (to authorize service deployment).

The evolution of functionality of autonomic operations with the processes and actions from the IT experts and the systems is presented by [Ganek and Corbi (2003)] under the form of 5 autonomic levels, similar to the adoption model's autonomic functionality axis (Figure 9). Described as an evolution, and not a revolution, the objective is that enterprises adopt autonomy not by reorganizing their systems but by introducing elements step by step such that autonomic functionality increases level by level (and not by jumping from the manual to the last autonomic level).

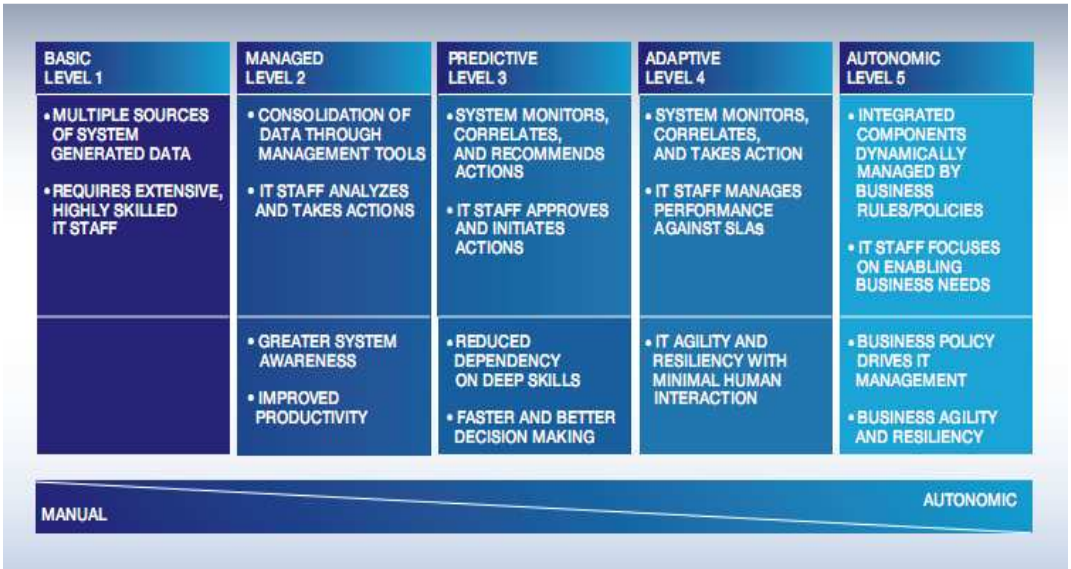


Figure 9 : Evolution of autonomic functionality [Ganek and Corbi (2003)]

Basic - The first level, fully manual represents the state of systems today, where the problematic of complexity is shown all the time. Each component is managed by highly trained and qualified IT experts, requiring a lot of human resources, with high costs, little

interaction between them and prone to human error. The human management also includes the monitoring and gathering of data.

Managed – At the next level, the adoption of autonomy is done towards the awareness of the state of the system and its components. The monitoring process is done in an autonomic way, gathering information and data from several sources and bringing them under centralized forms to the IT expert. This way, the time required by experts to synthesize the information is greatly reduced.

Predictive – At this level, after being able to monitor and gather data about its environment, a system is now able to also recognize certain configurations, predict configurations according to past evolutions and provide advice with what actions the human expert should take. Still, the system only raises alerts, it does not act by itself (the change is done manually). At this level, IT organizations are measured on the availability and performance of their business systems and their return over investment. In order to improve these, enterprises measure, analyze and manage transaction performance (by its critical nature). Predictive tools are used to project future measurement and performance and to make recommendations concerning future evolutions.

Adaptive – The fourth level of the autonomic evolution, goes further from the predictive level, enabling taking actions based on the monitored and analyzed data. Here, the system can also pick the best actions, and by implanting such a level of autonomy the human intervention is reduced to a minimum, the system being considered as 'agile'. At this level, system functioning is usually based on Service License Agreements. An SLA, is defined here, as a compact between a customer and a provider of an IT service that specifies the levels of availability, serviceability, performance, reporting, security or other attributes of a service, often established via negotiation from the two parts. It basically identifies the client's needs. IT organizations are measured on end-to-end business system response times, the degree of efficiency with which the system is used, and their ability to adapt to dynamic workloads.

Autonomic – The last level of the ladder, the system is fully autonomic, its operation being governed by business policies and objectives. Users themselves interact with the system to monitor the business process or modify its objectives. Looking back, from manual level where high skilled IT experts focused on system configuration and data gathering, here the staff can only focus on business needs, thus enabling fewer staff with lower technical skills. At this level IT organizations are measured on their ability to make business successful, and to improve their performance according to this objective. IT tools take into account the financial aspects that are part of the e-business activities. By advanced modeling of the system the e-business performance is improved which in turn allows to deploy optimized solutions with lower deployment times.

The autonomic computing adoption model allows both customers and vendors the integration of autonomic computing in flexible ways. Each one is free to decide up to which levels their system should climb, based on the needs, the costs and the specifications. Even if it is tempting to say that we design the system such it is on top of each axis of the adoption cube, this could easily be misleading, as we don't always need it. For instance,

while trying to obtain high autonomy with a data base for a commercial web site, the control scope can stop at the data base component level. There is no reason to extend it to the whole system of web design. There is always a question of balance and resource allocation (including costs) when deciding in which of the smaller cubes (from the axis intersection) a system is placed.

3.5 Autonomic computing architecture

Success in meeting the challenge of system complexity is based on two main factors. The first is the invention of new technologies, specifically the establishment of autonomic computing and self-management specifications. The second contains the architecture itself of the self-managed system, as it needs to fulfill and exploit the autonomic specifications appropriately. The right architecture is the key to providing the levels of autonomy within a system. Again, the objective is not to create and develop an advanced form of AI, giving computers human-like 'common-sense' or reaching human-level intelligence with them (or solving a problem of artificial intelligence). The goal is to help humans, by offering them an support with system management tasks. It is about evolving the systems so that humans can better manage them.

IBM's whitepaper [IBM (2006)] specifies three goals when creating an autonomic computing architecture:

- It must describe the external interfaces and behaviors required of individual system elements (at all levels)
- It must describe how to compose these elements so that they cooperate toward the goal of system-wide self-management.
- It must describe how to compose systems from these elements towards the objective of whole-system self-management (how do components interact, are organized, communicate etc.)

3.5.1 General architecture

The general architecture of an autonomic computing system is based on building several level blocks, which are then composed to assure self-management capabilities. The blocks interconnect via service bus patterns, allowing collaboration and communication using standard mechanisms, like web services. There are several elements that integrate with the service bus:

- *Manageability endpoints* – the standard manageability interfaces for managed resources (touch points)
- *Knowledge sources* – the sources of data and information that the system uses
- *Autonomic Managers* – the core entity of autonomic systems that implements the autonomic loop
- *Manual Managers* – the human management related part of the autonomic system

The composition of these blocks is shown, according to the IBM blueprint, in Figure 10.

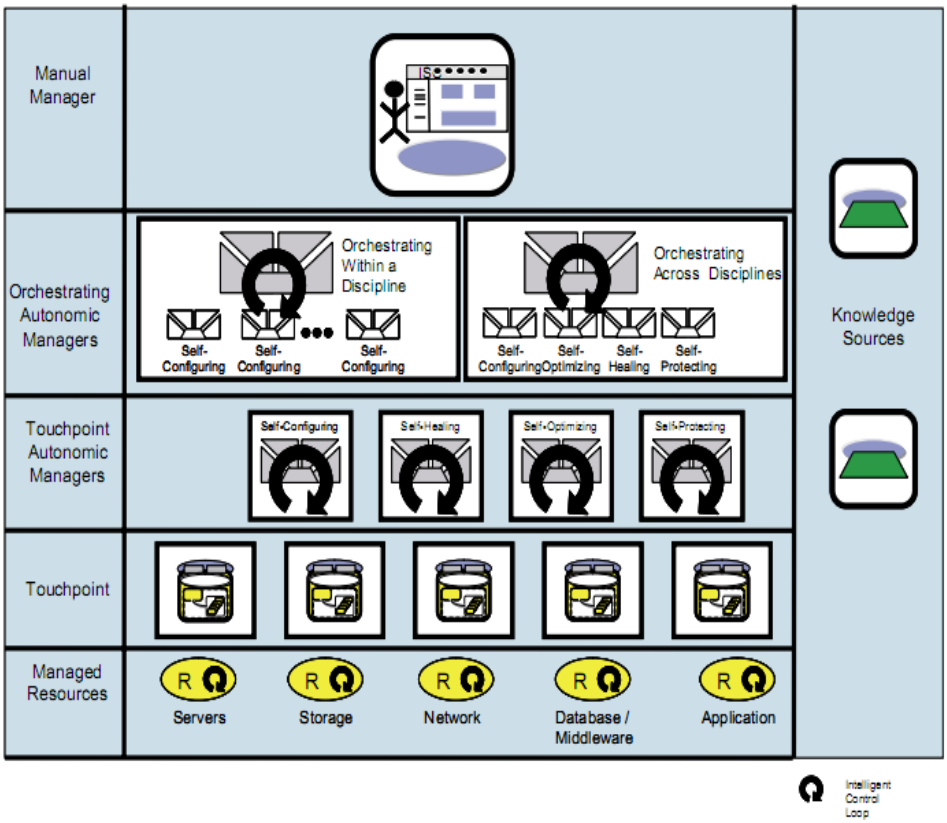


Figure 10 : The Autonomic Computing architecture [IBM (2006)]

The architecture presents five horizontal layers, which are all coupled to knowledge sources.

- *Managed Resources* – The bottom layer, which contains any component of the system that needs to be managed. These components (or resources) make up the IT infrastructure. They can be of any type, software or hardware, and even have embedded self-managing characteristics (i.e. a data base with certain intelligent management loops etc).
- *Touchpoint* – Consists of the management interfaces that are used to communicate with the resources. Touchpoints assure the communication between the resources and the superior levels of the autonomic computing architecture. Touchpoints must sense any modification or behavioral change with the managed elements and report it to the touchpoint autonomic managers in turn.
- *Touchpoint Autonomic Managers* – The layer the implements the autonomic functionalities via the intelligent control loops. Touchpoint autonomic managers are responsible for the implementation of the C.H.O.P. principles, by using a four state intelligent loop: monitor, analyze, plan and execute (MAPE-K loop).
- *Orchestrating Autonomic Managers* – The higher level of adoption for the CHOP principles, with orchestrating managers that regroup several touchpoint managers according to different areas of the system. They equally implement an overall intelligent control loop (the loop of loops).

- *Manual Manager* – The last level, which proves that autonomic computing adoption is done towards the evolution of current architectures. There is always the IT expert that will benefit from all the four layers underneath, and manage the system at the highest levels (business) using advanced and friendly user interfaces.
- *Knowledge Sources* – The element that relates to all of the five layers of the architecture, and which contains all the information and knowledge regarding the system's management. Knowledge sources are the central element of the architecture, and the information within relate to: administration policies, system topology, architectural organization, configuration parameters, symptoms and diagnostics, business rules and best practices, etc. The choice of implementation of the knowledge bases is not specified by IBM, as we can have several distributed sources or one central source. Either way, this is a dynamic component of the architecture, as, similar to maintaining any knowledge base, it must be up to date, synchronized and provide high availability.

The architecture provides a consistent view over configuring the computing environment on every aspect, from the technological components to the IT process services. ITIL describes the notion of a *Configuration Management DataBase (CMDB)* as a technology that tracks the configurable elements of a system and the relations that exist between them. Along with configuration and change management, CMDB is one of the fundamental elements in the evolution of autonomic systems, and several industry players provide their own implementations [IBM (2010)].

3.5.2 Managed resources

Managed resources make the first layer of the autonomic computing system architecture. These resources can be of any type, either hardware or software. Resources are the most basic element of an autonomic system and, in the same time they, are the base elements (as it is over these resources that autonomy is adopted). They are divided into several categories: [Normand (2007)]

- *Simple resources* – when the concerned elements are hardware (such as physical servers, routers, printers etc.) or software (applications, data bases, OLAP cubes etc.)
- *Combined homogenous resources* – or meta resources, that describe how the simple resources combine together into logical resources. For example, several OLAP cubes concerning archive processes can be grouped under a logical application of archive in a hierarchical way. An important aspect here is that combined resources are homogenous (meaning several resources of the same type are combined under a higher entity)
- *Combined heterogonous resources* – same as the category above, just that the combined resources are of different types.
- *Information System* – The whole IT system can be seen as a single resource too, similar to the combined resources. Actually it is a combined heterogeneous resource.

These categories are prior the one important aspect of managed resources. In the autonomic architecture they are presented as separate components, but usually they coexist and are related. Thus IBM doesn't specify in the blueprint how they are organized, a common way is to assemble them into hierarchical configurations.

3.5.3 Events

An autonomic system is in a state of continuous monitoring and gathering of information regarding its 'existence', and, according to specified rules and policies takes actions based on this information. Events are elaborated over this monitored information. According to IBM, there are two main types of events: timeless events and timed events.

Timeless events are events which don't depend on the time when they are triggered. One example is the filter pattern.

- *Filter Pattern* – the simplest of schemas, it verifies each event to see if it corresponds to the defined filter. If there is a match, one or more action rules are triggered (Figure 11)

Figure 11 : Simple filter pattern [Biazetti and Gajda (2005)]

Timed events are events which are time constrained. There are several patterns for timed events, such as:

- *Collection pattern* – events are collected periodically over a defined time interval. At the end of the period, the collected pattern is compared to the existing filters. (Figure 12)

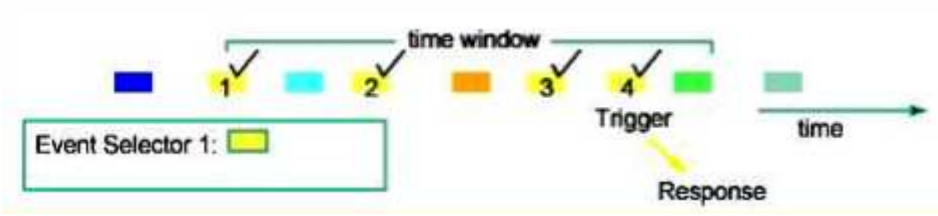


Figure 12 : Collection pattern filter [Biazetti and Gajda (2005)]

- *Duplicate pattern* – a schema specialized in identifying duplicated events. If there are several identical patterns during the defined time interval, only the first one (from the occurrence time point of view) is treated, the rest being ignored.
- *Threshold pattern* – represents a schema of states. As events are received thresholds are selected (based on static (min, max) or dynamic (average) values). During a defined time interval (fixed or gliding), these thresholds that are taken into consideration for the schema matching. (Figure 13).

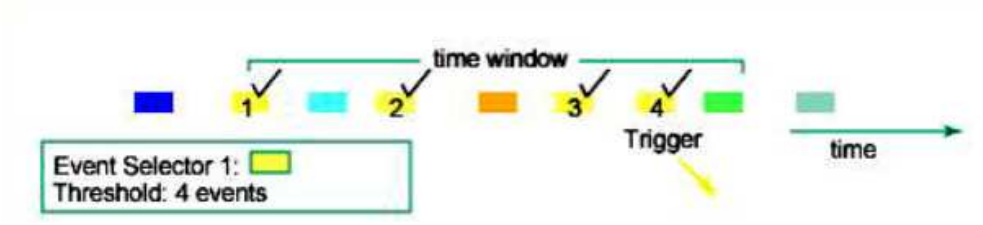
- 

Figure 13 - Threshold pattern filter [Biazetti and Gajda (2005)]

- *Sequence pattern* – a schema used for detecting the presence/absence of either regular or irregular sequences of events, over a defined time interval (Figure 14).

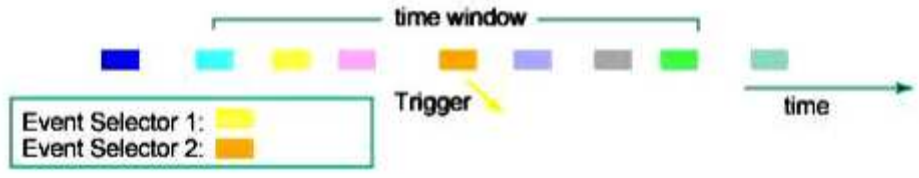
- 

Figure 14 : Sequence pattern filter [Biazetti and Gajda (2005)]

3.5.4 Manageability endpoints

Endpoints are the level of interconnection between the resources and the superior levels of the autonomic architecture, specifically the autonomic managers. A manageability endpoint (or touch point) exposes the state and management operations for the resources in the system. It communicates with the autonomic managers though the manageability interface. Figure 15 describes a touch point according to IBM's specification:

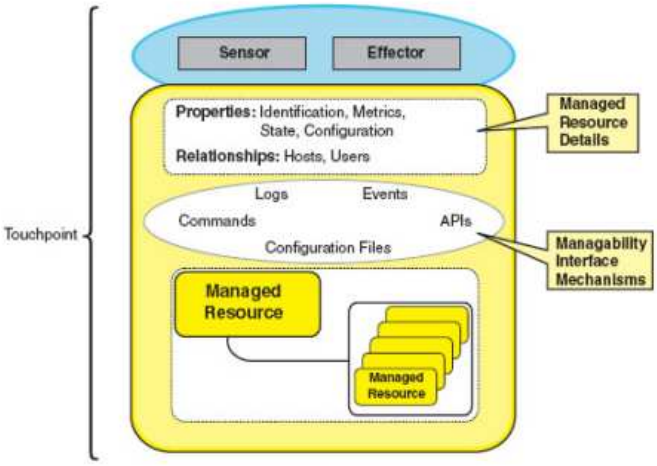


Figure 15 : Autonomic Computing manageability endpoint (touch point) [IBM (2006)]

Several components are described.

- *Sensors* – used to obtain the information and data from the resources. There are two states of functioning for a sensor: passive and active. A passive sensor will only

collect the information without taking any further action to make this information available. It will respond to information requests from the autonomic managers. An active sensor, besides collecting the data, will also be in charge of sending the data to the autonomic managers, without their special request.

- *Effectors* – used to perform operations on a resource. Effectors interact directly with the managed resources, and, similarly to sensors, have two operating modes: passive and active. In the passive mode the effectors act directly with given instructions from the autonomic manager. In active mode effectors use complementary information, such that they can trigger actions based on the information received.
- *Managed Resource* – the actual resource on which the sensors and the effectors act.
- *Managed Resource Properties* **–** all the information that describes the resource, its properties (parameters, state, configuration) and its relation with the others resources (servers, users etc.)
- *Manageability Interface Mechanisms* – contain the elements that are responsible for driving the interface. The use of standard manageability interfaces, such as the Web Services Distribution Management (WSDM)[IBM (2005a)], is one key aspect of the autonomic computing architecture.

The manageability interface mechanisms are further discussed because of their vital role in enabling autonomic computing. Advanced mechanisms for communication between the interfaces were founded along with the initiation of autonomic computing, based on either adopted existing standards or the development of new standards for exchange management. The need of inter element and resource communication, along with allowing autonomic managers to execute their functions (monitoring, configuration etc.) led to the development of the WSDM standard. It is ratified by the Organization of Advanced and Structured Information Standards (OASIS), providing a standard for a common interface for managed resources.

Further on, by collecting and aggregating log files information for use by the autonomic managers (especially for problem identification and system recovery), a standard was developed for event representation: *the WSDM event format (WEF)*, equally part of the OASIS standards. By comparison, normally a lot of time is spent by system administrators to look through hundreds of log files, in different formats, on different machines etc, and correlate the information manually. With the help of the WEF standard, a unified representation of these logs is done (natively or by conversion), which in addition can be used with autonomic managers. This results in a double advantage. First, having a unified form of representation, already gains the administrator valuable time from doing it himself. Second, by integration with the autonomic managers, even more time is gained as many of the tasks will be executed in an autonomous manner. An IBM survey carried in 2005 recorded up to 40% time reduction spent in problem determination and resolution, in enterprises which adopted the WEF standards. Moreover, this figure was the result of the adoption of the standard (first advantage), without any autonomic implication. In addition to the time reduction, it eliminated the critical runtime errors, improved the availability and the quality of services (the ultimate goal), and minimized the delays with client production schedules. On top of this, the implementation of autonomic managers further improves the system, allowing the functions to be executed via the intelligent loops.

Another standard for the manageability interface is the Solution Deployment Descriptor (SDD) [IBM (2008)]. The solution is focused on the self-awareness of software resources, as it enables software to identify its relations and dependencies within the system. This leads to understanding how elements are combined and more important allows the development of impact strategies (how change affects the systems). Self-awareness information facilitates installation validations, dependency checking, dynamic changes support and finally high availability. IBM surveys carried in 2005 showed that organizations using SDD technology realized up to 30% reduction in the time of solution deployment, while equally gaining up to 50% in the productivity of packaging staff.

Several elements are specified with the development of new standards, such as:

- *Symptom specification* – along patterns and schemas of events, symptoms should be specified in a common manner for problem determination or resolution activities
- *IT policies specification* – a unified format for representing IT policies and business objectives
- *Change request management* – consisted methods to treat changes within a system
- *Information consolidation* – a standard approach to obtain unified information, consolidated into a single CMDB

3.5.5 Autonomic computing managers

Autonomic computing managers form the third layer of the autonomic computing infrastructure. The autonomic computing manager is the core of the autonomic computing adoption, because it is responsible for automating the IT self management functions [Parshar and Hariri (2007)]. Basically it is an implementation that automates management functions and externalizes these functions according to the behaviors specified by the management interfaces [IBM (2006)]. It implements an intelligent control loop, as seen from the architecture, via which it collects the information concerning the system, analyzes it, elaborates plans based on the analysis output and executes the changes according to these plans. These four actions form the MAPE-K loop (Monitor, Analyze, Plan and Execute over a central Knowledge source). Figure 16 shows the building blocks of the autonomic computing manager according to IBMs specifications, with the connections to the manageability interface via the sensors and effectors:

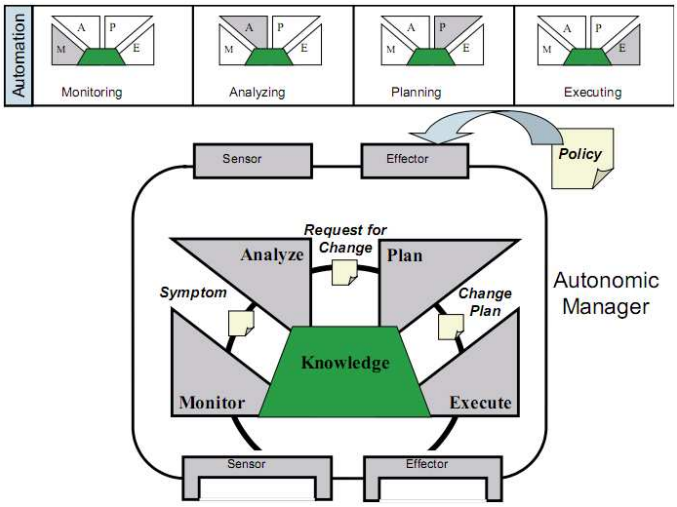


Figure 16 : Autonomic Computing Manager [Miller (2008)]

3.5.5.1 Monitor

The monitor function has the role of gathering, aggregating, correlating and filtering data from the managed resources, via the sensor manageability interface until a symptom that needs to be analyzed is recognized. Symptoms are constructed based on the monitored data and express the current state of the system. A symptom doesn't necessarily mean a problem as it may be misinterpreted. For example, a symptom would be the query response time when performing data retrieval operations over a data base, or the quantity of used RAM memory on a physical server. The elements that are observed during the monitoring phase include topology information, events, various metrics, configuration parameters etc. During the monitor state the manager can either be in a 'PUSH' or 'PULL' mode, according to the sensors. 'PUSH' means that the sensors are active, and they send information to the manager. 'PULL' means that the sensors are passive, and that the manager itself retrieves the information from the sensors.

Monitoring involves capturing all the elements describing the properties and characteristics of the environment (either physical or virtual). The sensors that are used to perform the actual operation of information retrieval can either be hardware (i.e. a temperature thermometer for measuring the CPU heat level) or software (i.e. the MS Windows Task Manager, for the resource usage with current processes). A first phase is to filter out the elements that are non-interesting for the autonomic manager. For example, it is pointless to measure the CPU temperature when following RAM allocation policies. This is why, most of the time the type of sensors and the monitored elements are resource specific. Accordingly, autonomic computing architectures identify two types of monitoring [Huebscher and McCann (2008)]:

- *Passive Monitoring* – as a monitoring that is done 'on-demand', usually performed by the IT expert. For instance a 'top' and 'vmstat' commands under Linux, which display the CPU and Memory usage statistics. Also under Linux, the 'proc' folder contains all the runtime system information that can be of interest for anyone trying

to manage the OS, such as: the hardware configuration of the machine, the mounted devices (media stores, disks, USB devices etc.), network interfaces etc.

- *Active Monitoring* –implies that the monitoring of resources is done autonomic at software level by the modification of the application's implementation. For example, when monitoring an OS, such modifications are meant to capture function or system calls that are able to return the interesting information. Active can be seen as an extension towards autonomy, as is no longer the IT expert that handles the task. As an example, the ProbeMeister [Object Services & Consulting (2002)] is able to insert probes into Java byte code (meaning application behavior can be modified during execution by the insertion of other java code).

One important question with monitoring is: from the sets of performance measures and tools, which are the most interesting for managing a dynamic environment? Interestingly [Zhang and Figueiredo (2006)] observed that only a small subset of available metrics provides 90% of the needs. The tendency is towards active and dynamic monitoring to facilitate autonomy. For instance, the proposition of QMON by [Agarwala et al.(2006)] consists in an autonomic monitor that adapts the monitoring frequency and data volumes, with the purpose of minimizing the continuous monitoring overhead while focusing on utility and performance data. It also points out that monitoring Service Level Agreements is an important factor for the quality of service and should always be considered while monitoring systems.

3.5.5.2 Analyze

This function has the role of analyzing the symptoms observed by the monitor state. For example a symptom that indicates an increase in the query response times of the OLAP bases on a server might be an indication of a need to increase in the servers performance (or the adding of supplementary servers) so that response times are kept within the required specifications. Based on the analysis outcome, which by analogy are the diagnostics, a set of plans are made for the next phase. The role of the analysis is also predictive. Based on the results from previous loops, the autonomic computing manager can predict future behaviors and suggest configuration modifications within the system in the analysis phase.

Analysis includes two ways of functioning, similar to the monitoring: passive and active.

- *Passive analysis* refers to diagnostic assessment. Based on the monitored symptoms a list of diagnostics is build that can be used further by the human expert. It is like going to the doctor and having a machine tell you the diagnostics, after which the doctor prescribes you with the medication. It is called passive analysis because there is no action proposed or taken, thus the MAPE-K loop is broken from the autonomic point of view. In this case we can speak of self-diagnostic.
- *Active analysis* adds action taking to the operation of diagnostic assessment. Alongside the list of diagnostics, a list of possible actions is integrated. Based on these actions, the MAPE-K loop works further with their planning and execution (of the ones that can be rendered autonomic, of course).

Analysis is crucial in the autonomic process, and is presented as the most important phase. Failure or incorrect data can lead to improper actions and even system destruction (e.g. an airplane trajectory analysis that instead of comparing meters with meters compares meters with feet).

3.5.5.3 Plan

Planning has the role of assuring a planning strategy for the actions send from the analyze state. Planning uses policy specification to build the plans, and it can vary from simple changes to a single resource to complex changes to multiple resources and resource groups. There are two possible scenarios for planning actions. Either the plans are already known, in the case which the role of the block is to choose the right plans and to send them to the execution. Or, either some of the plans are not known so strategies must be build according to the required modifications. For instance, let's consider a requirement for a restart of the logical server and the physical machine on which it is running. For stopping them first the logical server will be stopped, then the physical machine. When restarting, first the physical machine is turned on and then the logical server is started.

Planning involves, in the broadest sense, taking into consideration the data from the analyze phase, and translating it into the required changes for the managed resources. Sometimes overlooked, this phase actually is responsible for the outcome of the changes. Analysis indicates what to do, execution provides the actual execution mechanism but is planning that establishes how to do it. Consider the following example: on a server a restarting of services is required due to an application upgrade. As services depend one upon the other, the element of planning which indicates the order of starting them is vital. Failing at this means failing at reputing the system online [Stojanovic et al.(2004)].

There are several types of representing planning policies. The simplest would be the definition of Event-Condition-Action (ECA) rules, which act upon both analysis and planning. They can directly provide the plans from specific event combination. ECA policies take the form of "when event occurs and condition holds, then execute action". For instance, "if the average query response time for an OLAP base is over 5 seconds, and the amount of allocated RAM memory for this base is below 5% of its size, then increase the allocated RAM memory by 10% if its current value". There are several examples of such policy languages and applications in autonomic computing, such as [Lymberopoulos et al.(2003)]. One of the issues with ECA policies is that they scale very poorly. With large systems where there is a consistent growth in the number of policies, coherence between them becomes very hard to assure. Another limitation of the ECA approach are *stateless* configurations. A stateless configuration implies that there is no information on the state of the managed resource and the autonomic manager relies only on the imputed monitored data for analysis and planning.

Further, the authors of the study suggest that it is far better to render processes *stateful*, by keeping information on the managed resource on its state, which in turn can be updated progressively through sensor data and also can be used to enable reasoning. This led to a more complete state approach: keeping the state of the entire system along with the state of each managed resource, thus leading to the so called *architectural model-driven approach*.

An overall system is created by the autonomic manager, usually called the *architectural model*, and that reflects the environment, behavior, requirements and goals of the autonomic system. The model is in turn updated with the monitored data, and any decision or action that it proposed is bound to reflect the latest changes (thus avoiding synchronization problems). One of the main advantages of the model-based architecture approach comes from its mirroring capabilities. This way, a guarantee for a good outcome of a plan of changes is assured, by already successfully testing the changes on the mirrored model. Any failure detected in the mirrored model would stop the changes from taking place in the used model, while the used model would have never stopped working and providing services. Even if it can be interpreted that the use of the model-driven architecture is opposite to the use of ECA rules, it is not entirely true. Actually, the two share common points. For instance, any failure or 'healing' functions of the model would be represented as ECA rules (as the action to be taken or simply a rollback function).

Policy-Based adoption planning usually uses policies enabled via ECA rules, which determine the actions to be taken under certain conditions and events. Generally they are elaborated either by system administrators or sometimes translated from the technical documents. This is the simplest approach to elaborating planning strategies and is confronted with the problem of coherency. This is recalled by [Lupu and Solman (1997)], as an event my satisfy the conditions of two different ECA rules that run in parallel, and the outcome of one rule is an action that contradicts with the other satisfying rule. More difficult is the detection of such situations, as most of them arise at runtime (and not at conceptual level). In such cases a human expert intervention is required to modify the rules in the loop and solve the observed conflicts, each time. On the other hand, an architectural model is bond to represent a system, containing two elements: the system components and the component connectors. A system component is any managed resource, with no specification to the level of detail (be it an entire physical server machine or a simple data base). The connectors are the formalization of the connections between the components. The nature of these relations is not specified, therefore connectors can be seen in any direction Nevertheless, the architectural model does not describe a precise configuration, rather it offers a number of constraints and properties that a system must fulfill in order to be labeled as an implementation of the specific model.

One question of architectural model-based planning is, of course, how to represent the model, what language to use? The answer lays with *Architecture Description Languages (ADLs)*, such as Darwin or Acme. *Darwin* is one of the first ADLs, as a result of the work of [Magee et al.(1995)]. The formal representation is a directed graph in which the nodes represent the managed resources and components and the arcs represent the connectors between them. In this case the connectors describe the links between service provider and service demander components. The example given consists of a number of service offering servers and a number of service requiring clients. The idea is to guarantee that every client demand is satisfied within the threshold of the service levels. If the current number of servers is unable to provide these services, new servers are added in order to return the system to a valid model state. With this approach there is a copy of the architectural model in the knowledge source of each managed autonomic resource. By distribution of the managed system model, the system avoids the downfalls of problem detection and failure

with the entire system, which a central architectural model is prone to. The disadvantage of the distributed approach is keeping up-to-date all the copies from all the managed elements (usually achieved by autonomic broadcasts of the latest version of the model, implying that we can reach any component starting from any other component in the architecture graph). Another ADL is *Acme*. It is an adaptation framework that uses an architectural model that detects any need of adaption within a system [Garlan and Schmerl (2002)].

3.5.5.4 Execute

The fourth function of the autonomic manager is to execute the plans received from the planning phase. The changes which affect the managed resources are performed using the effectors. An important part of the execution is also to update the knowledge base with the latest changes to assure consistency.

Autonomic managers generally use policies, specifications, goals and objectives to drive their functioning. This means that the knowledge IT experts have on system management should equally be present in the knowledge base. The knowledge extends the capabilities of an autonomic manager, by specific knowledge types such as: new symptoms definition, new diagnostics etc.

In its current presentation form, the autonomic computing manager offers flexibility to the degree of autonomic adoption. System administrators can freely choose what functions and to what extent are they executed in an autonomous manner. For instance, one may choose to only implement the monitoring phase because it only needs raised alerts regarding the state of critical elements. The potential actions or changes are done only manually by the IT experts. Another scenario can be autonomic adoption over the planning and execution phases. The human expert specifies the modifications that must be made and leaves the autonomic system to plan them in the correct order and execute them accordingly.

3.5.6 Manual manager

The manual manager is an implementation of the User Interface (UI) that allows IT experts to manually perform some of the management functions and to supervise the entire autonomic process. Being on the top layer of the autonomic architecture, the manual manager collaborates with the orchestrated autonomic managers or with other manual managers (that is other IT human experts). It usually implies that there is always a human expert that interacts with it and no autonomic tasks are done at this level. A manual manager can also enable human experts to delegate management functions to the autonomic managers. User interfaces are very different and vary depending on the purpose of the management process. Still, there is an effort towards the usage of common console technology management, with the goal of creating a consistent human interface for IT infrastructure components.

Autonomic capabilities of self-management systems perform tasks that are either autonomic or delegated by the human experts, according to policies, goals and objectives. Certain tasks may be chosen deliberately to be human handled by the administrators, and the interaction between them and the system is greatly improved with the use of common

console frameworks that exploit industry standards and promote consistent presentation to IT professionals. The functions of these consoles are diverse, from simple setup operations to configuration operations and run-time monitoring and control. The role of the managing console is to help with tasks and activities related to service management and service delivery, by aggregating service related information. This information is extracted from various knowledge sources, such as: the CMDB, operational states, policies, goals and objectives. The aggregation operation is very important as it is based on the upper levels information with activities delegated by the human experts. As the level of autonomic adoption increases, so does the level of delegated tasks (passing from low to high level tasks).

The efficiency and client value of an integrated solution console is measured in the cost of ownership (by the level of efficiency with its administration), and the decrease of the learning curve when confronted with the addition of new elements to the environment (software products, hardware resources etc.). A better integrated console leads to gaining substantial time for the users (both administrators and clients) and avoids (or reduces to a minimum) the learning curves for new products. The common console architecture is based on standards (i.e. the Java API extensions JSR 168, JSR 127), and it can be reused and extended to offer any required functions, based on the expert's needs.

3.5.7 Knowledge source

The knowledge source (KS) or base (KB) is the core of the autonomic computing manager. It is presented as any implementation of a registry, dictionary, database or any other repository that provides access to the information according to the interfaces of the autonomic architecture [IBM (2006)]. However, the blueprint does not provide information of how the knowledge source is organized (single centralized source, distributed sources etc.) nor how the information is represented (database tables, ontologies etc.). Nevertheless, what is specified is that several aspects of the autonomic architecture should be contained in the KB, such as [Normand (2007)]:

- The topology of the concerned autonomic resource, such as the architectural organization or the hardware and software the resource uses. For instance, an OLAP server with several OLAP bases is described in the knowledge base with its resource consumption (RAM memory, CPU, storage capacity) and its links with the organization of the OLAP bases (which bases are for the archive, which are for reporting purposes etc.).
- Activity and logging of the concerned autonomic resource, meaning that all past events and activity is saved. This in turn is separated according to needs and specifications: monthly, trimestrial, archive etc.
- Management policies, meaning the totality of business policies, service level specifications, best practices, advice concerning resource allocation, parameter configuration etc. This type of information originates from multiple sources: readme documents, technical forums, IT professionals own experience and practice, etc.

IBMs description knowledge source points towards a collaborative direction, where different sources from different autonomic managers can communicate and aggregate over the hierarchical levels of the AC architecture. It basically extends the capabilities of an autonomic manager. An ACM can load knowledge from several sources and choose what is relevant and what should be ignored. This way, the specter of the possibly automated tasks increases over actions such as symptom pattern recognition or policy application.

Even if the knowledge source is presented as the heart of the autonomic computing manager, it impacts the entire AC architecture. This is why the existing literature expands its borders and speaks of knowledge base representation for the entire autonomic system. The organization of the knowledge sources within the whole knowledge base is left to the choice of the implementation of the AC adoption. Starting from the diversity of the knowledge available, the survey of [Huebscher and McCann (2008)] presents several methods for representing knowledge in autonomic systems, from which we remind:

- *Utility* – an abstract measure of the customer benefit. There are several ways of measuring utility, such as the amount of available resources or the quality and availability of a resource. For instance in an OLAP server where resource allocation is an issue between the OLAP bases, the utility is calculated as a mixture of the amount of allocated resources for each base and its financial costs, its importance and priority and the average query response and calculation times obtained with that allocation. Another example is in a resource provisioning system, where the utility is computed as a combination between the cost of allocated workload redistribution and the cost from the power consumption [Osogami et al.(2005)].
- *Reinforcement Learning* – is used to establish future actions and policies based on previous management activity. One of the most basic ways of reinforcement learning is to try different actions and observe their result and consequences [Sutton and Bart (1998)]. The advantage of reinforcement learning is that it doesn't require explicit models of the managed system, from where its use with autonomic computing adoption [Dowling et al.(2006)]. The disadvantage of reinforcement learning is its poor scalability with large systems, where there are a big number of states and the capability of learning from probing is seriously impacted. Because of this, several hybrid models have been proposed to reduce the time to converge or the number of states with the system [Tesauro et al.(2006)].
- *Bayesian Techniques* **–** Refer to probabilistic techniques for self-management, as actions and decisions are taken in autonomic ways based on probabilities and chances. In [Guo (2003)], it is shown how Bayesian networks are used for autonomic algorithm selection with the purpose of finding the optimal algorithms. Also, reinforcement learning feedback loops based on cost assumption has been used to attribute costs to self-healing equations for failure recovery, presented in ***[Littman et al.(2004)]***

3.6 Integration with artificial intelligence

Artificial Intelligence (AI) has a strong integration with autonomic computing, especially with the knowledge base and feedback aspects. AI can be used to the prior phase of transforming the non structured knowledge which drives the ACMs into structured knowledge that the ACM can understand (machine understandable format).

3.6.1 Concepts

The definition of Artificial Intelligence has been the object of numerous works from both scientists and philosophers. In the ancient Greece, Aristotel described the human being as a 'rational animal' and established the study of logic by formalizing the syllogisms, putting thus the bases of what is nowadays called intelligent reasoning [Mozes (1989)]. The objective of Aristotel's works in this area was to understand and explain how the human mind works (thinking and reasoning). A syllogism is a two-proposition logic, also known as the premises, that lead to a third proposition, also known as the conclusion. An example is shown below:

Table 2- A syllogism example

Minor Premise	All humans are mortals
Major Premise	All Greeks are humans
↓	
Conclusion	All Greeks are mortals

The first two are the premises, a minor and a major one, and are always considered as true, thus verify the outcome of the conclusion. This was the first formalization of the human deduction ability, and by extension of our knowledge. According to [Badiru and Cheung (2002)], it was much later on, in the 17th century, that new approaches were made in this area. An idea that our thinking and reasoning may be formalized under mathematical form was issues by Thomas Hobbes, also known as the grandfather of IA. He concluded that if a machine is capable of doing complex mathematical operations, then it should be equally able to duplicate and imitate the human thinking behavior.

Yet, the real advancement with IA was yet to come in the 19th century, when the bases of the first computing machines were developed. The English mathematician Charles Babbage layed the foundations of a calculus machine, under the influence of Blaise Pascal's and Gottfired Leibitz's works. In 1833, he was working on a programmable analytical machine which was able to take decisions (the traditionally IF-THEN-ELSE) similar to a nowadays computer. Another English mathematician, Georges Boole, formulates the bases of today binary logic, corresponding to the two states: true or false, 1 or 0. It was the birth of the digital logic. [Alesso and Smith (2006)] indicates that in 1931 Kurt Gödel has formulated the incompletion theory, that was a major contribution to the mathematical (and afterwards the computing) worlds. His theory shows that for certain problems there is no solution, thus creating spaces of unresolved problems. Gödel is also called the father of computing as we know it today. In 1938, Claude Shanon makes a proposition towards using boolean logic in a machine (or computer). Further on in 1945, an American (of Hungarian

origins) physician and mathematician, John von Neumann, proposes the fact that (now well under development) computing machines should be capable of more than rendering predefined operations and processes. He proposes the implementation of simple logic to allow machines choose their 'execution paths' from several alternatives. It is Alan Turin that will consolidate the bases of the AI, and is also known as its father. The concept of Turing machine combined with the Von Neumann concepts leads to the development of the first programmable machine: the computer. We can see in Figure 17 the evolution of AI throughout history from Hobbes to the creation of the computer.

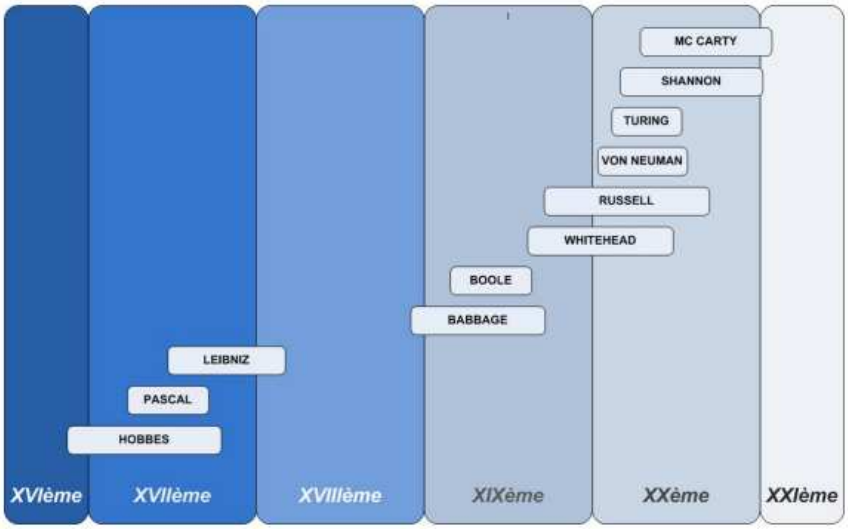


Figure 17- The history of AI [Normand (2007)]

In order to compute the level of 'intelligence' of a machine, Turing has developed a test, also known as the 'Turing test'. The performance of an intelligent machine was computed by its capacity of handling a normal human conversation with another human partner. The human would state facts, ask questions and try to get answers from the machines, whereas the machine would try to respond in a fashionably manner to all the requests, asking in turn questions and performing reasoning on the learned facts. A recent example of such behavior was made with the *Artificial Linguistic Internet Computer Entity (ALICE)* [A.L.I.C.E. (2010)]. ALICE makes use of the LISP functional programming language. Functional languages allow the formalization of concepts, and were developed form the early 50s, with the inventor of the LISP language John McCarthy. Other similar functional languages are Prolog or Camel.

Today, AI is part of several domains, which lead to its development and application cases. Among these, we mention:

- *Natural Language Procession (NLP)*: Is mainly used in text mining for understating the meaning of words and sentences (semantics). Some of its applications are automated text translation, grammar and semantics analyzers, orthographical assistants or event voice recognition.
- *Visual Intelligence* – used especially for form and visual pattern recognition (such as face recognition); recent advancements use AI to regain eye vision.

- *Robotics* – construction of more complex robots with automatic movement gestures and sensors, reaction and feedback to the environment etc.
- *Learning* – a more sophisticated domain, split into supervised and non-supervised learning. Supervised learning is based on existing knowledge and case scenarios. Non-supervised learning bases on its own feedback process, thus it must 'create' the knowledge.
- *Reasoning* – describes two research areas: deduction and pro-activeness. In the first case, based on a series of given facts and using IA one can deduce a list of conclusions. With pro-activeness, an intelligent system, in addition from the deduction, will act based on these conclusions.
- *Knowledge Engineering* – related to the NLP, this research area of IA relates to all aspects of knowledge transformation from non-structured (in any form) to structured machine understandable format.

3.6.2 Multi agent systems and autonomic computing

In the context of autonomic computing systems, a parallel is done with *Multi-Agent System (MAS)*. A MAS is a system composed of a suite of intelligent parts (agents) that exchange messages between them in order to help solve complex problems [Wooldridge (2009)]. A MAS is usually used in situation where a single monolithic structure is not enough to solve the problems. The advantage of a MAS is that it allows the resolution of very difficult problems, on distributed systems, which otherwise would have not been possible. The disadvantage is that the architecture is much harder to implement and maintain.

There are several characteristics of a MAS, the most important being:

- *Decentralization* – there is no 'mothership' or single controlling entity that follows the functioning of the system.
- *Autonomy* – agents are semi or fully autonomous. Semi-autonomous agents function in dependence with results from other agents, while fully autonomous agents are capable of running all by themselves without any other interface.
- *No entire visibility* – no agent has full visibility over the entire agent system. The area of visibility of an agent is proportional to its communicating agents.

The application of MAS with autonomic computing is explored by [Tesauro et al.(2004)] with the Unity software architecture, where the authors show the similarities and some of the advantages and disadvantages using the two. The traditional IBM adoption model is around a central management point, whereas the Unity architecture aims at achieving self-management of distributed autonomic systems. What is noticeable with this approach is that each autonomic element is "goal-driven self-assembly". This is described as each autonomic element knows in start only a high-level description of its goal (e.g. be accessible only 24/7) and the contact information for this (in the so called registry). An entire process, from registry contact to service identification and specific roles follows such that the managed element is able to identify its specific tasks related to the overall goal.

An architecture for self optimization for a data center scenario is shown in Figure 18. It implies the existence of a shared resource point, under the usage of several application environments with the goal of optimizing the resource allocation for each of the environments.

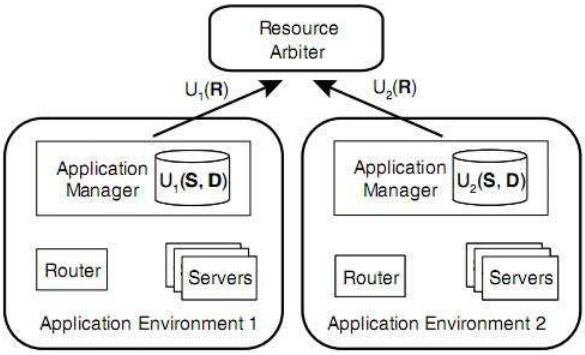


Figure 18 : Self optimization architecture for a data center scenario [Tesauro et al.(2004)]

U is the service utility function for each application, and it depends on *S*, the level of service for the environment, and *D*, the demand for services in the environment. As this is a constant changing function, there is a constant need to balance the resource load such as that the sums of *U* is maximal. This is the role of the resource arbiter, which allocates resources based on each ones demand and its service levels. Each application manager disposes of an agent that modifies the application parameters, requests or freezes resources and communicates with the arbiter in this process. The approach is easily scalable, suitable for a dynamic environment, where applications come and go. The arbiter has an autonomic computing manager that decides through the loop the new resource allocation, while taking into consideration the messages from each application autonomic manager.

3.7 Conclusion

Autonomic Computing was presented in this chapter as a possible solution for the IS management key problem: system complexity. We have presented the main autonomic computing concepts, with the four autonomic purposes (the CHOP) and the AC adoption model for the different systems and levels of automation. Next, the core of the autonomic model, the autonomic computing manager, was detailed, as the entity who implements AC via a closed intelligent loop: the MAPE-K. We have studied how AC has already been applied, and how the field of Artificial Intelligence and the Multi Agent Systems provide an alternative approach of autonomic systems.

The question that we were trying to answer, as a link with the previous chapter, is if autonomic computing is suitable for IS management, for helping the IT experts with those low level tasks by passing them to the machine. The common applications such as IBMs DB2 optimizer [Markl et al.(2003)] or the Microsoft SQL Server ‘black-box’ [Mateen et al.(2008)] show the orientation of the industry towards task automation, and reinforce the fact that solutions such as autonomic computing are viable alternatives.

As presented from the autonomic manager description, the heart of the ACM, and subsequently the entire AC architecture, are knowledge bases. IBM specifies their existence, but doesn't impose a certain format or standard of knowledge representation. This is why in the next chapter we explore the AC knowledge management aspect and the representation of the knowledge bases with the help of semantic web technologies, namely ontologies.

4 Knowledge Management

"A little knowledge that acts is worth infinitely more than much knowledge that is idle."

K. Gibran

4.1 Introduction

This part revolves around all of the elements presented so far, as it focuses on the knowledge engineering aspects: acquisition, transformation, formalization etc. Knowledge engineering is a very vast subject and covers all areas where information treatment is required with information systems. Everything from text documents, data base records to web sources, ontologies and ultimately human knowledge is encapsulated by this research domain. This chapter comes as a logical continuation for the previously described items, and specifically the core of AC adoptions.

The organization of the chapter focuses on a specific form of representing knowledge (which is used with our approach): ontologies. We equally present the work that was done with ontologies in collaboration with DSS and AC. This is very important as it was the starting point of this thesis.

4.2 The semantic web

One of the forms of implementing knowledge bases is web semantics (ontologies). As this is the technology that we use with our approach, we make a detailed state of the art in this section to understand where it started from, its actual state and future directions for its development.

Web semantics is based on the *World Wide Web (WWW)*. The WWW is an enormous source of knowledge, size being its greatest power and its greatest weakness. Power because if we search long enough we might find just about everything about anything. Weakness because much of this mass of information is neither structured nor organized. Sometimes we search for hours to find the required piece of information, sometimes we won't find it at all, sometimes it is contradictory etc. So, face to this challenge, web semantics tries to bring order into chaos. Tim Berners-Lee describes the semantic web as a 'web of data that can be processed directly and indirectly by machines'. [Bizer et al.(2009a)] empower this view over the importance of web semantics and focuses on the term of linked data, with the references and best practices of publishing and connecting data on the web.

4.2.1 Semantic web concepts

The first attempts towards linked data dates from 1945, when [Bush (1945)] described the 'memex', a machine that was able to manage (create and update) links between documents, existing in the form of micro magnetic tape. From this work, later in 1965 Ted Nelson [Nelson (2010)] put the bases of the notions of "hypertext" and "hypermedia", to further work with Andries van Dam to develop the first hypertext editing system at Brown

University in 1968. The same year, finishing a work started in 1962, Douglas Engelbart presents the 'Mother of all Demos', the first public demonstration of a hypertext interface as part of the NLS system (an oNLine System for linking hypertext).

The major milestone contribution (and the father of today's web) was made about 20 years ago, when Tim Berners-Lee presented an internet hyper-based initiative for global information sharing, while he was working at CERN [CERN (2010)] [Berners-Lee (1989)]. In this first article, he focuses on expressing ways of linking information systems, with all of their elements: people, groups of people, software modules, documents, hardware etc. It was the first step towards linking data that didn't belong to the same specific domain, but was required to work together. The article also emphasizes the introduction of hyper text and common keywords.

In their book, [Alesso and Smith (2006)] link the birth of the web with the moment at which Berners-Lee introduced the first notions and protocols: URLs, HTTP and HTML. These are the three main bricks for what the web is today.

URL (Universal Resource Locator) permitted to link each document with a unique location on a network, allowing the localization of HTML documents in that era. Nowadays an URL is used to identify any web page, every site having its own unique URL (e.g. http://www.sp2.fr). Having a unique reference for each page permits to link them with one another and eliminates any ambiguities when accessing them. The format of the URL is described by the RFC 3986 [NetworkWorkingGroup (2005)].

HTML (HyperText Markup Language) is a simple form of representing and encoding documents, in a formatted way. It has been developed from the SGML *(Standard Generalized Markup Language)* presented in 1969 by Charles Goldfarb, project manager at IBM**.** SGML is the first formalized markup language and was used for exchanging documents between printers. One of the key points with HTML is that it allows not only the formatting of text when interpreted and displayed by a web client, but also allows users to click on hyperlinks pointing towards any resource (other http pages, images, sounds, videos etc.)

HTTP (HyperText Transfer Protocol) is as the first (and current) protocol for collaborative, distributed hypermedia systems, according to the RFC 2616 [NetworkWorkingGroup (1999)]. It is a simple basic protocol for assuring client-server requests (e.g. a web browser as a client requires responses from a web page stored on a server). A series of status codes are used to indicate the functioning of the protocol, the most known being: 302 Found, 403 Forbidden, 404 Not Found.

In 1994, based on the first web bricks, Berners-Lee puts the foundation of the *W3C (World Wide Web Consortium)* [W3C (2010e)]. It is the main international standards organization for the WWW, in charge of maintaining standards such as HTML, XHTML, XML, RDF, SOAP etc. Any new format has to pass thorough five stages before reaching the W3C recommendation label. These are: working draft (WD), last call working draft, candidate recommendation (CR), proposed recommendation (PR) and W3C recommendation (REC). Yet, due tot the constant evolution of standards, a W3C recommendation is not final, having the possibility to reach a new version of the

recommendation if sufficient matter has been introduced to make the difference (e.g.HTML 1.0, HTML 1.1, HTML 2.0).

With the creation of the W3C consortium, the growth of the web and advances with this technology were exponential. If back in 2003, [Daconta et al.(2003)] evocates more than 3 billions of documents with 500 million of users, nowadays, in 2010, [WorldWideWebSize.com (2010)] estimates over 21 billion pages with more than 1,8 billion internet users (26% of the worlds population).

4.2.2 Actual state

[Berners-Lee (2006)] outlined a set of rules for publishing data on the web in a way that this data becomes part of a single global data space. These are known as the 'Linked Data principles', and are listed below:

- The usage *of URIs (Universal Resource Identifiers)* as a name for 'anything'. In analogy with the URL, URIs form the bigger class. If URLs allow the unique description of web pages, URIs allow the unique description of all things providing a unique identifier for each existing concept (e.g. book, author, house, dog, www.sp2.fr). The picture below shows this relation:

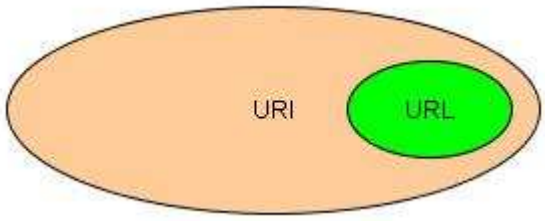


Figure 19 : The URI URL relation

- Use HTTP URIs, such that the resources are available to everyone. For example, if we use the URI *CarFordFocus* to describe the 'ford focus car' concept, it can work fine locally only, but it will not be available to others. A *http://www.mycarexample.com#FordFocus* URI will allow the identification of all ford focus cars via the common http link, so there is no ambiguity.
- Use W3C standards when looking up and recovering existing URIs. For example, using the *Rich Description Format (RDF)* to describe the URIs and the links between them, or the *SPARQL (RDF query language)* to retrieve URI information. This allows the definition of new URIs which refer to concepts already described by existing URIs.
- Last, URIs should not be isolated, but should exist with regard to other URIs, thus relations between URIs is a must (e.g. *http://www.mycarexample.com#FordFocus is a http://www.mycarexample.com#car*).

Still, even with these rules of describing web data, finding the required data is always a challenging task, as we don't know where and how to look for it. This is where search

engines develop, with both syntactic (majority) and semantic capabilities, such as google, yahoo or bing. Even a new branch of meta search engines are developing, that permit parallel searches on several search engines at the same time. To keep the information up to date and the web 'actual' softbots are used to scan and index permanently the web pages.

We have mentioned the terms syntactic and semantic, when speaking of data retrieval over the web. Most of the current search engines use syntactic values, with some meta information via key words associated to the pages. Each page has a list of associated tags, and the inverse selection is done when recovering a list of web pages corresponding to a set of given key words. This process works well, but sometimes can be tricky. The catch is that it never provides with the information, but rather with the web pages potentially containing the information. There is no relation between these keywords, and sometimes search results can completely 'miss' the target (e.g. a search for *SP2 Solutions* will return results from the Microsoft Windows XP SP2 package rather than the SP2 Solutions company). Nevertheless, current search engines such as Google [Google (2010)] adopt sophisticated algorithms including smart indexation, linking tags or page ranking to identify the potentially good results.

Concerning the semantics aspect, engines such as PowerSet [PowerSet (2010)], Hakia [Hakia (2010)] or Swoogle [Swoogle (2010)] offer some semantics capabilities, relating the search queries to the existing links between the corresponding URIs. Swoogle is specially dedicated to searching over ontologies, thus over well formatted data that respects the four rules of linked data. Some of them may even be capable of responding to semantic questions such as: *How old is Angelina Jolie?*, but these aspects are far more complex and get into the fields of natural language processing and semantic comprehension. Moreover, these search engines are yet in their incubation stage, thus demanding constant need of advancement in this direction.

One may ask: why speak so much of the web? The answer is simple: because all information is out there. We try to manage an IS - we need all the information required to do so, we need to adopt autonomic computing – we need the most suitable form of representing the knowledge bases. Knowledge representation on the web is the key, thus the relation is direct and essential.

4.2.3 Limitations

The current state of the web and the distributed knowledge has several weak points, some of which will be improved with larger adoption of web semantics and linked data practices. In the actual context of this thesis, these weak points should be seen as equivalent problems with managing IS.

Quantity of data The most common problem of information management is the quantity of information available. The number of web pages increases greatly each year, thus any approach of treating all data is constrained by its mass.

With the adoption of linked data strategies, some of the information can be reduced at the interesting semantic data only, offering significant reduction in the terms to search and facilitating the indexing of the new pages.

Information completion the second major problem derives from the quantity of data and concerns the completion of the information with regards to the interest of the one requesting it. Completion refers to the aspect of finding all the pertinent required information. When searching and managing best practices for managing a decision support system, we expect that the information concerns only DSS and not other types of IS. This is not always the case, the page ranking algorithms analogy working in the same way. Some piece of information that almost nobody has mentioned (even if it may be critical) will likely be skipped when searching. At the other end, a piece of information that is very common and that everybody is aware of will always be brought first. This is logical from the search point of view: one may not build the chimney of a house without the foundation, roof etc., but without the chimney the house is unlivable as it will fill it with smoke from the fireplace.

[Triou et al.(2007)] present a rapport of 1 to 500 between what is called the visible web and the invisible web. The visible web is the part that is indexed and referenced with a high page rank by the search engines (what everybody knows) and the invisible web is the part with very few references (what only few people know).

By increasing the number of links between the web pages, according to the linked data principles, we greatly increase the chances that the information concerning with our needs will be recovered.

Search limitation – when asking for information sometimes we are faced with a ridiculous number of results. A google search for 'decision support system management' brings over 14 millions results. It is impossible even to just click open each link (1s per click = 162 days 24h a day 7days a week), not even speaking of analyzing the content within these pages. [Groison (2001)] shows that that 90% of the users don't search after the second page of search engines, and that 95% only make single word queries. Thus the pertinence is well reduced and limiting the number of responses is very difficult. With web semantics search limitation is hard to achieve, due to the high number of direct related topics. This can be reduced working over information quality.

Information quality can be seen as the pertinence of the information that is brought to the user when searching. By allowing the expression of more complex requests (i.e. the combination of 'OR' or 'AND' keywords or the use of special characters such as "). These 'artifacts' permit in the same times the reduction of the number of results and potentially increases their pertinence. Nevertheless, these facilities are only used by 5% of the users, from the same study of [Groison (2001)].

The real problem with quality is context awareness of the search engines, and this existing problem is the lock to the evolution of information management today. The key is web semantics.

***Example**:* the search of 'Decision Support Systems management' returns over 14 million results, but with no semantic implications. For instance, a technical article with the title: Improving Cache Allocations with Essbase Data Warehouses will never be found with this search, although the issue is very well known to DSS experts. It is a specification of a generalization: Essbase is a specification of the general term Decision Support Systems. Without the link specified, the search engine will not detect it.

Web semantics and linked data come help solve the example above, by creating links between the Essbase software to the general Decision Support Systems concept. This is by far the most important advance with web semantics, as they permit both topology links (generalization – specification) and inter concept links.

General data management can be seen as the sum of all the above problems. It is the user's experience with searching the needed information. There are neither clear guidelines nor methodologies when searching for it. Web semantics and ontologies bring some answers, but opened issues still remain.

4.3 Ontology

The concept of ontology has existed from human antiquity. It comes from the Greek terms *onto,* which means *being*, and *logos,* meaning science. Thus ontology means the science of being, and was introduced by Aristotle [Aristotel (350 BCE)]. Throughout the ages the usage of ontologies flourished within the philosophical world.

The evolution of information system brought ontologies to the attention of the IT experts, as a very powerful way of expressing knowledge, especially over the WWW.

Definition. In 1992 T. Grubber defined an ontology, for IT, as a specification of a conceptualization, 'a description (like a formal specification of a program) of the concepts and the relationships that can exist for an agent or a community of agents' [Gruber (1992)]. He points out that their main purpose is to enable knowledge sharing and reuse. An updated definition of the ontology is given over 15 years later, for the specific domain of computer science by [Liu and Özsu (2008)]. Here, an ontology 'defines a set of representational primitives with which to model a domain knowledge or discourse'.

4.3.1 Ontology concepts

The development of an ontology passes through several stages of evolution: controlled vocabulary, thesauri, taxonomy and the complete structure of an ontology. This evolution is shown in Figure 20:

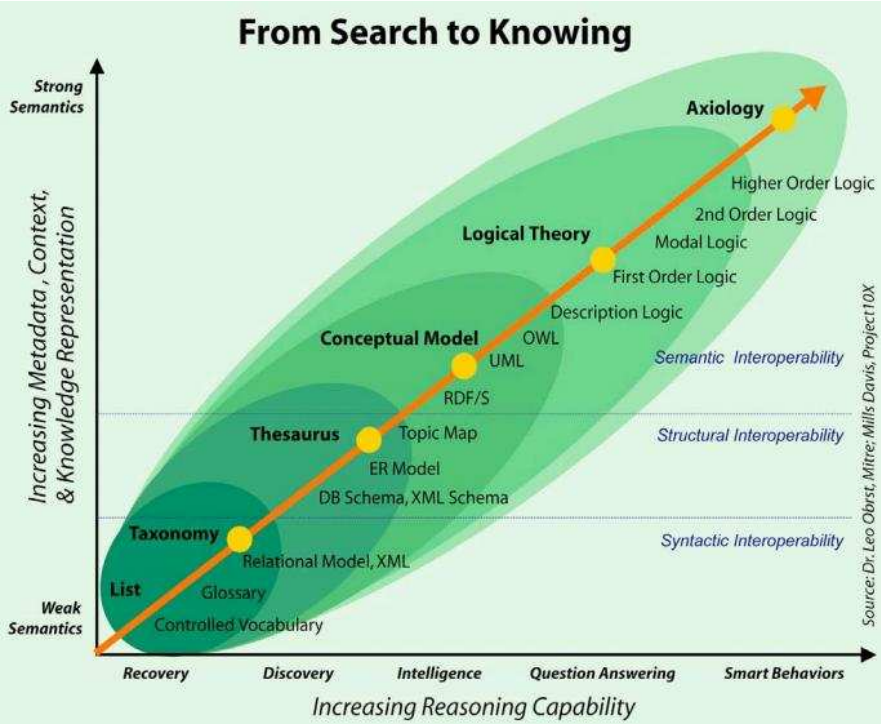


Figure 20 : The evolution of ontologies [Lassila and McGuinness (2001)], [Davis (2006)]

The controlled vocabulary is the first abstraction towards an ontology representation. A controlled vocabulary provides with a basic way of expressing interoperability between several domains, thus describing a set of common words and concepts that are generally available (e.g. a physical server means the actual machine in both a hardware ontology and a software ontology). Actually, its definition is a list of terms, where a term is a particular name for a concept. The strictness of this definition is given by the vocabulary registration authority, with two general rules that must be respected [Pidcock (2003)]:

- If there is a term with the same name for several different concepts, then its name must explicitly solve the ambiguity.
- If the same concept is referenced by several terms, then one of the term is chosen as principal, whereas the others are defined as synonyms of this term.

Example. Consider the following two phrases: *The cache allocations for the cube are not optimal and I really enjoyed watching the Cube*. The word cube in the two phrases doesn't reference the same concept. In the first phrase it indicates a specification of a *data mart* whereas in the second it indicates a *movie*, with no relation whatsoever between the two. The other way around, when speaking of specification of data marts, we can consider that the terms: *multidimensional base* and *OLAP cube* are synonyms.

Glossary and Thesauri It's the second advancement towards ontology. In addition to the controlled vocabulary, the terms from a glossary have meanings associated to them. These meanings are described in several ways, usually in human natural language for easier understanding.

A thesaurus is an advanced dictionary with semantic relations between the terms, such as: synonyms, antonyms, homonyms etc. There is no explicit definition of hierarchies, but certain of the term links provide the deduction of generalization-specification. An example of thesauri is WordNet [Miller (2010)], a large lexical database for the English language. According to the latest statistics, v3.0 of WordNet contains almost 206941 concepts.

Taxonomy is an explicit hierarchical organization of a glossary. Taxonomies were introduced in the 18th century by Carl Linnaeus [Linnaeus (1731)], with a study over plant hierarchies. The main element of the taxonomy is the generalization-specification relation: *isA.*

Example. The image below shows a very simple the taxonomy of humans.

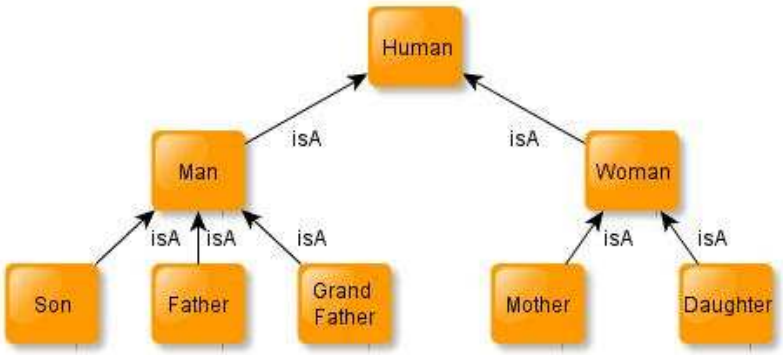


Figure 21 : Human taxonomy example

There is the overall generalization class, *Humans*, with a division between *Man* and *Woman*, (*a Man is a Human, a Woman is a Human*). Underneath, the hierarchy expresses concepts regarding the organization of a family (*a Son is a Man, a Mother is a Woman* etc.). In addition of the direct tree parent-child relation, the *isA* relation propagates on all the levels of the hierarchy tree. Thus, *a Father is a Man* and *a Father is a Human*.

Ontology the top level of the evolution, an ontology develops the taxonomy by allowing inter concept links, not only hierarchical links.

Example. In the taxonomy example, the only arrow properties were the *isA*. The figure below indicates some new links that transform the taxonomy into an ontology. We can see the existence of two new properties *isFatherOf* and *isMarreidTo* which link concepts on the same level of the hierarchy. *A Father is a father of Son* in this direction only. *A Father isMarriedTo a Mother*, and this works in the opposite direction too as indicated by the arrows, *a Mother isMarriedTo a Father.*

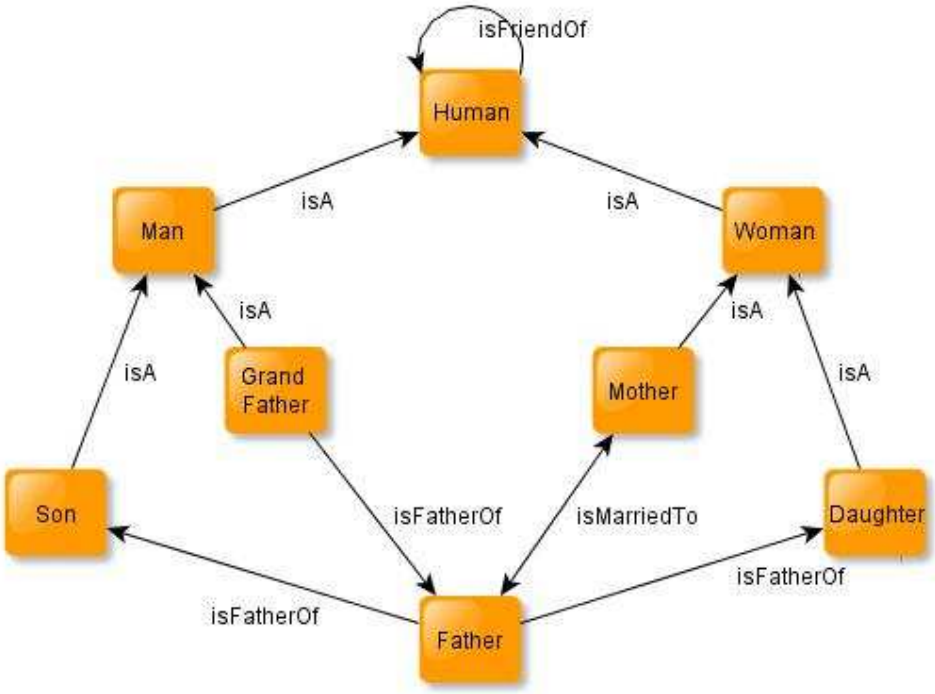


Figure 22 : From taxonomy to ontology example

4.3.2 Structure

Let us reconsider the definition of the ontology: *the formal, explicit specification of a shared conceptualization* [Gruber (1993b)], [Gruber (1993a)]. Each of the elements in the definition indicates a specific part of the structure of an ontology, and impacts its architecture.

- *Formal* indicates that the ontology should be in a standard (ideally machine readable) format that anyone is able to understand.
- *Explicit* means that the concepts in the ontology must be described each and every one explicitly. Implicit relation deductions are enabled via inference engines, but the concepts must be explicitly described.
- *A conceptualization* represents an abstract model of a domain, thus the concepts that are used to describe a domain.
- *Specification* shows that an ontology should is able to instantiate the general conceptualization of the domain, and thus the specification is a complete description of the domain at a given moment.
- Last, *shared* is the key element, indicating that the ontology should be available to everyone and everyone should be able to bring its contribution to it. Therefore an ontology express group common knowledge and not individual single knowledge (it can but it is not its purpose).

The elements from the definition above indicate clearly that there is no correct or best approach when building an ontology, as it is a community effort. The ontology model

depends on the purpose, objectives and type of domain and application we try to build, and this proves to much to handle from start. The process of building an onotology is somewhat comparative with a living being (as it develops itself), thus being continuously predicted towards change and modification. All these aspects are well emphasized by [Noy and McGuinness (2001)] in their guide of starting developing ontologies.

The elements that built an ontology are split into three categories: concepts, relations (properties) and axioms (rules). From [Pretorius (2004)] the components of an ontology are organized in:

- *L* – the lexical input for ontology concepts and relations, defined as string sets
- *C* – the totality of concepts that describe the specified domain
- *H* – the taxonomy of the concepts from C (the *isA* relation)
- *R* – the relations that exist (apart the *isA* taxonomy relation) in the specified domain
- The relations that link the elements from L with the concepts from C and relations from R.
- *A* – the list of axioms or rules used to describe different constraints and deduction elements.
- *I* – the specification of the concepts (also known as the group of individuals)

The central component is the *concept*. Relating to [Bouaud et al.(1995)], a concept is defined by one term, used to describe an object in intention or in extension. Intention means describing the object by using the relating properties (e.g. *John is very intelligent)*. Extension means describing the concept by specifying its rapport with the generalization (*John is a Father*). From this point of view there are three relations between concepts: *subsume, part of* and *instance*.

The *subsume* relation represents the base of taxonomies, describing hierarchical structures. A concept C1 subsumes another concept C2 if C2 has all the properties and characteristics of C1. For example, *Man* subsumes *Father*, as all characteristics of a father are characteristics for a Man. The *Part of* relation describes aggregation over the concepts. For instance, SingleChidlFather or TwinFather indicate that they are a type of Father, and together aggregate as Father. Finally, *instance* expresses the instantiation of the concept (the leaf of the tree), such as *John Doe is an instance Father*, where John Doe is the actual person.

Properties are characteristics of the concepts in the ontology, and are defined over *a domain* and *a range.* The domain indicates which concepts have the respective property, whereas the range describes with which concepts the initial concepts are linked (by the property). For example, the *isFatherOf* has the domain as *Father* and the range as *Son* and *Daugther*.

Properties allow concept distinction and help eliminate ambiguities through *defined* and *un-defined* concepts. A defined concept implies the existence of a set of properties (sufficient constraints) that permit the identification of concept specifications. Alongside there are the necessary constraints that are compulsory properties the concept must have, but don't suffice to identify the concept specification. To continue with our example, the *Father*

is specified by the *isFatherOf* property which restricts that: for a concept to be a father it needs to have at least one link with the *Son* class via the *isFatherOf* property.

One important aspect of structuring and defining properties within in an ontology is the *'property of properties'*, a property being defined as having several types [Gruber (1993b)]:

- *Symmetry* – if concept C1 is related to concept C2 by property P, then C2 is linked to C1 by property P. E.g. the *isMarriedTo* property: *John is married to Mary then Mary is married to John*.
- *Transitivity* – if concept C1 is related to concept C2 by property P, and concept C2 is linked to concept C3 by property P then C1 is linked to C3 by P. E.g. the *isFriendOf* property: *John is a friend of Mary, Mary is a Friend of Jack then John is a friend of Jack*.
- *Functionality* - a functional property can only have single value at a time. E.g. the *isMarriedTo* property, as a person can only be married to one person at a given moment.

Last, the definition of *axioms or rules* permit the transformation of implicit in explicit facts with the ontology. [Staab and Maedche (2000)] show the importance of using axioms and rules, especially with large scale ontologies. In their turn, axioms are organized by their role (e.g. the hierarchical relations: *Father is a Man* indicates an axiom over the *isA* property). Rules in turn are declared as X -> Y where X represents a list of conditions and Y represents a list of facts that are deduced from those conditions. For example, *if John is the father of Jack and Jack is the father of Joe, then John is the grandfather of Joe.* This allows the 'population' of the *GrandFather* concept.

The complexity of the ontology concepts indicate that there is a need for at least some guidelines for building ontologies. Therefore, T. Gruber in [Gruber (1993b)] distinguishes a list of evaluation criteria for ontologies:

- *Clarity/Completeness* implies that the ontology should 'speak for itself', thus should be clear to anyone who uses it. In addition, an ontology should be complete form the definition point of view so there are no undefined concepts from the domain it represents.
- *Coherence* implies concepts and axioms are logical and are not denying one another. It is also a measure of evaluating the quality of the ontology.
- *Extendibility* describes the level of 'standardization' of an ontology. This translates by how easy it is for others to reuse it and/or extend it by adding new concepts, modifying axioms etc. This reinforces the collaborative aspect of ontologies. In most cases, the creation of a new ontology from scratch is useless, as there are existing ontologies that can be reused.
- *Hierarchy granularity* refers the level of detail that an ontology has with its taxonomies. The deeper the taxonomies are, the higher the granularity is and the easier it is to add new concepts and reuse existing ones.
- *Minimum semantic distance* between the children that are siblings (share the same parent), expresses that similar concepts should be kept together. One guideline is that a parent should have no more than twenty children.

4.3.3 The web pyramid

The web pyramid or the semantic web layer cake was introduced by Tim Berners Lee with the W3C. Its purpose is to describe the various layers and steps towards the formalization of data over the web, allowing it to become semantic and, ideally, trustworthy. The web pyramid is shown in Figure 23.

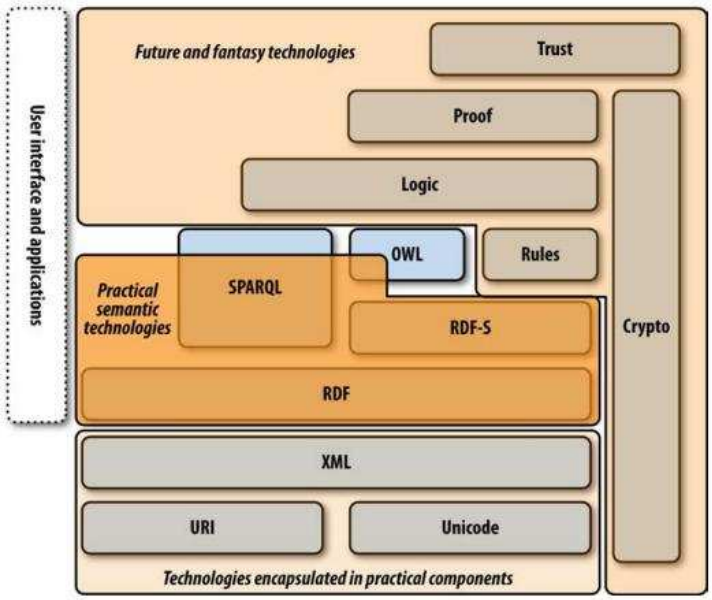


Figure 23 : Semantic Web stack or layer cake [Segaran et al.(2009)]

The evolution of the levels in the layer cake is comparable with the evolution of the knowledge representation formalisms. It starts from the most basic levels, from the first URI concept formalism and goes all the way up to the proof and trust levels. We detail them with regards to the technologies used by our approach (ontologies).

URI and Unicode were the first steps towards the formalization of the web. As shown earlier in the semantic web introduction, the URI (and URL) were the first meta-data that allowed the unique identification of a concept.

XML – *XML (eXtensible Markup Language)* [W3C (2010f)] is the first structured form of representing knowledge on the web. It derivated from SGML in order to suit the web knowledge representation needs. XML is standard for information transfer and document specification, and is still highly used today (i.e. SOAP or RPC [W3Schools (2010)] Web Services are XML derived and exchanged messages are in xml format).

The drawback of XML is that it represents purely data structures, there is no intelligence associated with the represented elements and there is no notion of semantic data with simple XML. For the restriction of the data structures and the format of various XML documents, *XML Schemas or DTD (Document Type Definitions)* are used. This provides constraints over the content of the XML document.

Example. An XML representation DTD for the human kind implies the presence of the xml tags of the elements human, man or woman etc. The DTD will also constraint each man or women to have an age and a name property, as seen below:

```
<human>
    <man hasAge="30" hasName="JohnDoe"/> <!- valid -->
    <woman hasAge="28" /> <!- non valid as it requires the hasName property-->
</human>
```

XML Query [W3C (2010g)] is a querying language which facilitates data retrieval from XML documents (but also databases, remote web documents etc).

RDF (Resource Data Format) is the language that introduces semantic capabilities. An overview of RDF is shown by [Grobe (2009)], with the purpose of solving the XML semantic 'handicap'. In [Wang et al.(2005)] RDF is described simplistically as sets of resources that connect to other resources through properties. Resources and properties are identified as anything that has an URI or a value assigned.

RDF is the first ontology description language; sentences used for the description of the domain are known as *statements* and are expressed under the form of *triplets (subject, predicate, object).* The three correspond to the elements of a simple sentence (subject, predicate and object) , and RDF sentences can be expressed graphically, from where the nomenclature of *RDF Graph*. An example of a graph describing a person is shown in Figure 24. It is composed of two statements (triplets) *(John Doe, isFatherOf, Jack Doe)* and *(John Doe, isMarriedTo, Mary Doe).*

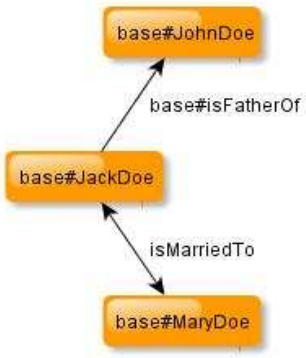


Figure 24 : RDF simple graph example

Again, similar to the XML, simple RDF poses no restriction over the data that is used to build the sentences. Therefore, in analogy with the XML Schema, *RDFS (RDF Schema)* [W3C (2010c)] was introduced, allowing specific vocabularies to be used with RDF sentences. RDFS is an extension to RDF and it allows the expression of taxonomies as its biggest advantage over RDF, with the introduction of the properties *subClassOf* and *subPropertyOf*. Another major RDFS contribution is the introduction of domain and range definition for properties, including the major data types (char, string, int, date etc.). In addition, a list of specific vocabulary properties are added, such as the *seeAlso* property allowing the reference of semantically related concepts or the *label* allowing an equivalent name (not unique) for the object.

The drawback of RDFS is that it lacks the representation of axioms and rules and their support with reasoning (inference). Improvements to RDFS were proposed to try to overcome this, such as [Delteil et al.(2001)]. Also, a proposition for an interchange format *Ontology Interchange Language (OIL)* [Fensel et al.(2001)] was made in order to help the development of RDFS, by permitting description logic methods for increased expressivity.

Ontology (RDF-S, OWL, SPARQL, Rules) – is the superior level of expressivity in rapport with the RDFS, by the introduction of *OWL (Web Ontology Language)* [W3C (2010b)]. OWL derived from RDFS by adding several capabilities, most notable axiom and rule inference and inference engine capabilities. Nowadays OWL developed to its 2.0 version with very high expressivity, thus opening the door to new possibilities for developing and working with ontologies. The usage of *Rules* in the next cake layer is the natural extension that makes OWL so powerful. On the other hand standards as SPARQL are present for data retrieval from ontologies.

Figure 25 shows a comparison of the evolution of knowledge representation forms (with OWL on top). RDF is at the bottom with its capability of representing a semantic network, then RDFS with frame capabilities and finally OWL with the power of description logic.

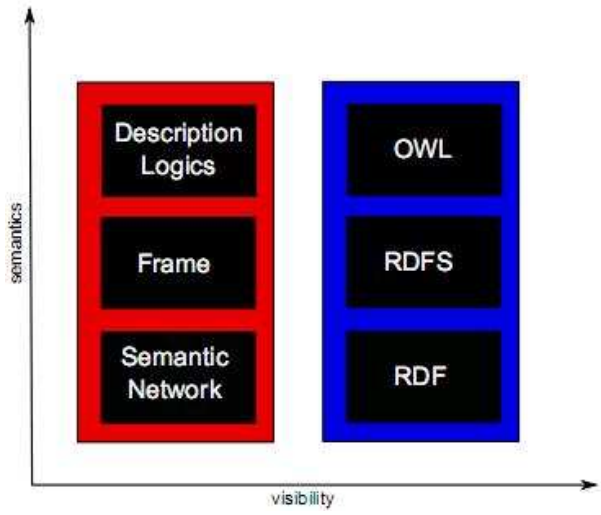


Figure 25 : Evolution of the knowledge representation forms adapted from [Ding et al.(2005)]

A more detailed discussion over ontologies is done in the next sub section, dedicated to OWL and its technologies.

Proof and Trust is the last layer of the pyramid, indicating the 'ultimate challenge' for knowledge formalization which is the proof and the levels of trust. With the expansion of semantic web and usage of linked data, the natural question is: how much of this data is valid, where does it come from, can we trust it? As [Segaran et al.(2009)] express it, working with real world data is full of surprises. One may end up false, incomplete or completely useless data. The advice is always to best know the knowledge source: *blindly*

crawling through linked data is an exciting way to program, you never know what you will get.

With the introduction of standardization through the W3C, some trust problems can be resolved. The W3C a reliable source of information. Examples of well known trusted vocabularies are: RDFS, FOAF [Mika et al.(2005)] or Dublin Core [Dekkers and Weibel (2003)].

4.3.4 OWL – web ontology language

OWL is the most cmplete language of describing knowledge in the current web semantics context, using the advantages of RDF and RDFS and bringing several improvements, as recalled by [Horrocks et al.(2003)], such as:

- The declaration of property types: symmetry, transitivity, inverse or functionality
- The declaration of complex classes as a result of combined logical operators: union, intersection, disjuncture
- The classification of individuals to classes depending on their properties, thus restricting the direct instantiation process. This identifies the defined and not defined classes. There are two types of restrictions: necessary and sufficient (defined class) and necessary.
- The support of declaration of axioms and rules and the connection with inference engines, generating new sentences from the existing ones.

The main elements of the OWL vocabulary which permit the identification of the basic concepts (class, individual, property) are shown in the Figure 26 (adapted from [Antoniou and Harmelen (2003)]):

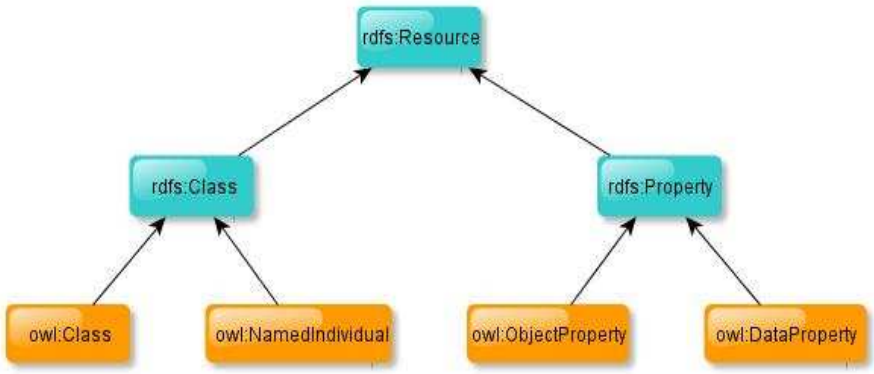


Figure 26 : OWL main concepts derived from RDF adapted from [W3C (2010b)]

- *owl:Class* - used to describe the hierarchical taxonomies.
- *owl:NamedIndividual* - the class instances, which are derived from the generic rdfs resource.
- Two types of properties, both used for class restriction definition and concept linking.

- *owl:DatatypeProperty* which is used to link individuals with literal types (string, integer, char etc.)
- *owl:ObjectPropery* – expressing links between individuals and classes and other ontology individuals/classes

The relations and declaration of concepts with OWL led to the definition of three sublanguages, with the version 1, the separation criteria being expressivity and the application scenario [Grau et al.(2008), W3C (2010b)]:

Figure 27 : OWL 1 Sub-languages adapted from [W3C (2010a)]

- ***OWL Lite*** – the simplest form, suitable for beginning working with OWL and aimed at reducing the computational complexity. The language lacks the inference and advanced classification support, but it compensates with simplicity and fast computation times.
- ***OWL D(escription) L(ogic)*** – provides the maximum expressivity while keeping reasoning and inference capabilities with axioms and rules. It includes all OWL constructs, remains computational accessible from the point of complexity and is recommended for advanced ontology description. With our approach, we use the OWL DL logic and the OWL 2 QL (see next paragraph) profile.
- ***OWL Full*** – provides the maximum expressivity (such as linking individuals to classes by object type properties). Basically anything can be expressed. Nevertheless this comes with the cost of decidability, thus having ambiguous sentences. Moreover, inference engines don't provide support for OWL Full ontologies, thus no reasoning capabilities are available.

With the advancement of OWL, OWL 2 presents three different profiles (derived from the previous separation) as described by [Grau et al.(2008)] and [W3C (2010a)]

- ***OWL 2 EL*** – based on the EL++ family of description logic, its main purpose is to assure the efficient reasoning with large ontologies, thus performance is chosen over expressiveness. OWL 2 EL ontologies are suitable for classification (class-subclass and individual-class belonging inference).
- ***OWL 2 QL*** – based on the DL-Lite family of description logic, are suitable for conjunctive query answering in efficient time (logspace). It is used when large ontologies must be queried, but with simple data schemas (lightweight ontologies). Most of the time there is a passage through relational database representations of the ontologies.

- ***OWL 2 RL*** – designed for advanced reasoning with rules through forward-chaining rule system. It is suitable for lightweight ontologies with large number of individuals, thus the necessity of operating directly with the RDF data sets.

With our approach we have used the OWL DL sublanguage. The list of OWL DL constructors is shown in Figure 28, where *A* indicates the classes, *C* the description of classes, *o* represents individuals, *R* object properties, *T* datatype properties, *B* data types, *D* data range, *v* is a value and *n* is a non-negative integer.

Classes	
A	
intersectionOf($C_1 \dots C_n$)	$C_1 \sqcap \dots \sqcap C_n$
unionOf($C_1 \dots C_n$)	$C_1 \sqcup \dots \sqcup C_n$
complementOf(C)	$\neg C$
oneOf($o_1 \dots o_n$)	$\{o_1\} \sqcup \dots \sqcup \{o_n\}$
restriction R	
someValuesFrom(C)	$\exists R.C$
allValuesFrom(C)	$\forall R.C$
hasValue(o)	$R : o$
minCardinality(n)	$\geq n\ R$
maxCardinality(n)	$\leq n\ R$
cardinality(n)	$=\ n\ R$
restriction T	
someValuesFrom(D)	$\exists T.D$
allValuesFrom(D)	$\forall T.D$
hasValue(o)	$T : o$
minCardinality(n)	$\geq n\ T$
maxCardinality(n)	$\leq n\ T$
cardinality(n)	$=\ n\ T$
Data Range	
B	
oneOf($v_1 \dots v_n$)	$\{v_1\} \sqcup \dots \sqcup \{v_n\}$

Figure 28 : OWL DL constructors adapted from [W3C (2010b)]

We notice the presence of restriction on both types of properties (object and datatype). OWL 2 supports several advanced restrictions to the datatype properties, one of the most notable being the capability of mathematical equality expressiosn over datatype property values (e.g restriction T with *greaterThan(n), lessThan(n)*) which was first introduced with OWL 1.1.

For a complete example of an ontology we recommend the 'famous' pizza ontology described by [Noy and McGuinness (2001)], which allows the reader to go through all the steps of *creating your first ontology*.

4.3.5 Rules and inference engines

The complexity of OWL DL arises questions regarding how modifications affect the expressivity, completeness or consistency of ontologies. In order to answer them, reasoning

and inference engines were proposed, such as Pellet [Sirin et al.(2007)], Racer [Haarslev and Möller (2001)], FaCT [Horrocks (1999)] or FaCT++ [Tsarkov and Horrocks (2006)]. An inference engine is defined as a software program able of deduce new facts or knowledge from known facts by using inference rules.

Inference engines take into consideration several points when reasoning over ontologies:

- *Subsumption* – the general-specific relation between concepts, used for taxonomy and class subclass classification (e.g. *Man subsums Father*).
- *Equivalence* – identifies two concepts as being conceptually the same (e.g. *Man* equivalent with *HumanMale*).
- *Instantiation* –identifies an individual as a member of certain classes depending on its properties and the class restriction axioms (e.g. *John Doe* is a *Father* because he has an *isFatherOf* relation value).
- *Correctness* – expressing the intentions of domain experts
- *Minimally Redundancy* – reduces to a minimum the usage of non desirable synonyms.
- *Rule chaining* – allows the creation of new knowledge from the existing knowledge by the specification of rules. Inference engines are able to decide the dependence between a group of rules (i.e. results of a rule are used as premises for other rule). Moreover, they are capable of deciding the consistency impact of running a set of rules over an ontology.

Inference engines open the doors to an intensive rule usage. The authors of [Patel-Schneider (2007)] describe the impact of rules, and how certain sets of rules can be ontology safe, starting from DL-Safe rules [Motika et al.(2005)] and implementing with SWRL rules [O'Connor et al.(2008)]. Several questions for expressing OWL rules are presented by the authors, from which we mention:

- *Horn or disjunction rules ?*– starting with simple Horn clauses leads to a conservative state for the ontology
- *N-ary or unary and binary predicates* – by cause of maintainability and implementation, unary and binary predicate rules are the best way to express rules with OWL.
- *Dataytpe variables* – the integration of data type variables with OWL rules due to the specific owl:DatatypeProperty, enabling mathematical expressiveness.
- *DL-safe or weak-safe?* – the preference is towards DL-safe that doesn't violate the concept constraints, thus keeping a consistent state of the ontology.

Rule semantics and syntax are offer different study areas. From the point of view of syntax, there are several standardization attempts for rule description, such as: SWRL, RIF, Jena or SPARQLUpdate.

SWRL (Semantic Web Rule Language) is the first extension for OWL for rule expression. An application environment SWRL API is presented by [O'Connor et al.(2008)] and presents tools such as the SWRL Tab [O'Connor (2008)] for OWL IDEs. SWRL was

intended to be the semantic web rule language, including high level abstract syntax for Horn clauses.

Characteristic to SWRL is that rules are expressed in terms of OWL concepts, are saved as a part of the ontology and reasoning with them is supported by the major inference engines. Several SWRL extensions were developed, such as the SWRLB which permits the expression of more complex rules including mathematical expressions, string operations, argument binding, or ELP [Krotzsch et al.(2008)].

Example. All humans which have the name starting with *Nic* and their age is greater or equal than 18 are considered to be adults.

Human(?x) ^
hasName(?x, ?name_of_x) ^ swrlb:startsWith(?name_of_x, "Nic") ^
hasAge(?x, ?age_of_x) ^ swrlb:greaterThanOrEqual (?age_of_x, 18)
→
Adult(?x)

One of the disadvantages of SWRL implementations is the capability of explicitly selecting the order of rule execution, as rules come with the ontology and are executed as a whole in a single passage.

RIF (Rule Interchange Format) was intended to be the world wide standard for rule expression with information systems (knowledge base, expert etc.). There are two approaches with RIF: RDF-compatible and the first order logic. RDF-RIF is a working draft (June 2010) [W3C (2010d)] and its mapping is over RDF graphs. The expression of a rule is in XML syntax, similar to the XML transformation of a SWRL rule. The first order logic presents the rule in a Prolog style.

Example: In a first order logic the previous example rule would be:

Forall(?x, (Adult(?x) :- And(Human(?x) age(?x, "15"^^xsd:integer) name(?x, ?name) startsWith(?name, "Nic"^^xsd:string))))

RIF is suitable for RDF, but it lacks the extensive OWL support.

Jena Rules are supported by the rule engine used with the Jena ontology interface [Grobe (2009)]. Jena rules are expressed over the RDF/OWL graphs, and are described using sentences (triplets) from the ontology and existent built-ins [Jena (2010)]. The format of the rules is very similar to the SWRL rules.

The advantages of Jena rules is that they are implemented apart from the ontology (i.e. text files), which allows a better control over the order of execution of rules, their representation and their storage. Moreover the Jena custom built-ins allow the definition your own built-ins, thus extending the expression capabilities of the rules.

Example. The same Adult rule expressed with Jena is:

(?x rdf:type :Human)
(?x :hasAge ?age_of_x) greaterThan(?age_of_x, "18"^^xsd:integer)

(?x :hasName ?name_of_x) regex(?name_of_x, "\bNic\w")*
→
(?x rdf:type :Adult)

The disadvantages of Jena are that it is a specific format, based on the Jena Rule Engine, and that only Jena implementations can make use of these rules.

SPARQL Update and future version of SPARQL. SPARQL is the W3C recommendation for the semantic web query language (over RDF/OWL graphs). Its main purpose is the efficient querying of OWL DL ontologies, with subsets such as the SPARQL-DL [Sirin and Parsia (2007)].

The drawback of SPARQL is that it assures data retrieval only, not allowing modifications to the inquired graphs. Therefore, extensions such as ARQ and SPARQL Update were developed, allowing the modifications (deletion, insertion) of triplets with the triplet stores. The main advantage is that these operations are possible both on the asserted and the inferred model. The future version of SPARQL, 2.0, is supposed to support these natively, a moment at which we can call a true equivalence between SPARQL for web semantics and SQL for relational data. An inquiry on how SPARQLUpdate is used to modify relational data is shown in [Hert et al.(2010)], with the similarities between the two technologies.

Example. the Adult rule expressed with SPARQL Update:

```
INSERT
{
        ?x rdf:type :Adult.
}
WHERE
{
    ?x rdf:type :Human.
    ?x :hasAge ?age_of_x.
    ?x :hasName ?name_of_x.
    FILTER (?age_of_x >= 18 && REGEX(?name_of_x, "\bNic\w*", "i")
}
```

We notice high similarity with the SQL syntax. Even if initially SPARQL was not designed for rule implementation, it can offer a good alternative for new users. Nevertheless it doesn't rely on an inference engine; all rule control is left to the one declaring the rules (no rule checking, no chaining engine etc.)

A disscusion here is required over the meaning of rules.As rules expressed under the same syntactic form may have different meanings, and that is where semantics play their role. Unifying the ssemantics of rule is difficult and is usually applied to a very specific area or domain. Such an example, for gene regularitory networks, is shown by the authors of [Agier et al.(2007)] who propose a framework for unifying rule semantics, for this particular domain.

4.3.6 Tools and applications

This subsection presents an overview of the main technology adoptions of ontologies, along with some very popular tools used for ontology development. Most of the applications are from the research field, but some have managed to penetrate the industrial world, indicating the potential of the semantic web for business.

Oracle 11g this version of Oracle introduces a semantic module, with support for RDF and inference capabilities for RDFS and OWL. Data querying is done both with SQL and SPARQL syntax (via the SEM_MATCH keyword) [Das (2009)]. This is the first major editor adoption of the semantic technologies, showing their potential with large scale implementations. A set of best practices for working with RDFS and OWL is provided by [Oracle (2009)], such as tuning considerations, RDFS rule bases, user defined rules and inference benchmarks for large scale ontologies.

A guide for developing semantic web applications with Oracle is given by [Wu (2007)]. It shows that the semantic capabilities were already present in the 10gR2 version, and how the 11 version developed them further for a scalable solution of managing millions of triplets with: the introduction of semantically operators, support for OWL sub languages, partial DL semantics or the Jena model Oracle adaptor.

TopBraid Suite – initially an open source project, TopBraid is now a professional commercial reference for working with ontologies, developed by TopQuadrant. [TopQuadrant (2009)]. It points out Top Braids purpose towards bridging the gap between the business collaboration needs and technology, through semantic products and services.

Sesame is an open source Java framework used for querying and storing RDF data. It was initially developed by Aduna for an EU research project On-To-Knowledge. It is presented as the choice for exemplification by [Segaran et al.(2009)] due to its administration interface, usage simplicity and strong performance.

OpenLinkVirtuoso is a very high scalable object-relational database system with support for both relational SQL and RDF, SPARQL operations [TopQuadrant (2004)]. It is produced and sustained by OpenLink, who plays an important role in the Linked Open Data community. One important application of Virtuoso is ***DBpedia*** , the ‘wikipedia’ of linked data. The DBpedia community and the efforts and publications around it prove the interest of linked data adoption for web semantics, currently counting over 4,7 billion pieces of linked information, from fields like: geography, entertainment industry, social networks and scientific publications [Bizer et al.(2009b)].

Freebase similar to the open link virtuoso, the freebase database allows the integration of linked data. Users can add data and link to other data building a web of connections. Danny Hills from Metaweb said, speaking about freebase, that ‘it is more valuable to operate across multiple systems, opening up silos of data and creating interconnections between them’ [Farber (2007)]. Thus, the purpose of freebase is to build all these links between the data, from multiple data sources, actually counting more than 10 million topics, 3000 types and 30.000 properties [Freebase (last accesed 2010)].

Jena is an open source Java framework for working with RDF/OWL. It is developed by the Hewlett Packard Semantic Web Research, and has a strong community around it. The main advantage of the Jena API is its complexity and freedom of working with the semantic technologies [Grobe (2009)]. Several aspects make Jena the ideal choice for experienced and professional users:

- Full built-in support for both RDF and OWL syntax
- Integrated inference and rule based reasoners (with direct connection to existing reasoners such as Pellet). There is also a Jena internal inference engine that provides a powerful alternative for deducing knowledge.
- SPARQL and SPARQL Update/ARQ query support. The strong development community around it led to the implementation of ARQ and SPARQL/Update support to extend the capabilities of working with ontology models.
- Two high-performance persistent solutions for storing ontologies,
    - *TDB* – the triplet graphs are stored in several indexed files on the disk, providing a fast access and high performances. This is advised for local accesses.
    - *SDB* - the ontology models are stored in relational DBs, thus the graphs are stored in data bases either locally or remotely. It has the advantages of portability, but with slower performances.

On the other hand, its greatest power is also its greatest weakness, as the complexity Jena provides requires a steep learning curve and solid knowledge over several technologies from the third party providers [Segaran et al.(2009)]. Another weakness is that for reasoning is done only with models loaded in the RAM memory [Oracle (2009)], thus restricting inference capabilities for large scale ontologies. In the same context of large ontologies, it is advised that form a certain point (> 5000 concepts), it is best to pass through a relational DB representation of the ontology (once all the inferences are made), to greatly improve the information retrieval process (SPARQL vs SQL) [Necib and Freytag (2005)].

KAON (Karlsruhe Ontology and Semantic Web framework) is an open source framework for working with onotlogies, developed by the University of Karlsruhe. It was introduced in 2002 [Bozsak et al.(2002)] as an academically research product, and provided as a visualization and development tool for many authors [Stojanovic et al.(2004)], [Bozsak et al.(2002), Maedche et al.(2003)],[Noy and McGuinness (2001)]. It was presented with the objective of efficiently working with large scale ontologies.

It provides a backend KAON Server for persistent ontology storage and access, a KAON API for application development and an interface for working with web services. Its objective is to provide ontology interfaces for web-based semantics-driven e-services. KAON is based on RDF, the server being an RDF application server.

Protégé both an open source API and a knowledge acquisition and management tool, it is maybe the most used tool throughout academics. It is build by the Stanford University School of Medicine for medical purposes [of Medicine University of Standford (2010)]. Highly supported by a large community, it provides several features such as:

- Native support for working with the all the known ontology supported syntax: RDF, OWL, N-triple, Manchester etc.
- Plug-in development for assuring high visualization capabilities of represented knowledge, such as OWLViz or Jambalaya [CHISEL (2008)]. Visualization is a very important aspect of working with ontologies, subject treated by [Krivov et al.(2007)], with an overview of the main tools and frameworks that serve this purpose.
- Capability of storing ontologies by connection to DBs (support for the main DBs types: SQL, mySQL, Oracle, DB2) or by several file formats (XML based RDF/OWL, n-triple).
- Support for graphically working with SWRL rules, with the SWRL tab [O'Connor (2008)].
- Native support for reasoning engines such as Pellet, FaCT or Jess [Wang et al.(2004)] and possibility of connection to any external reasoner.
- Collaborative development of ontologies, enforcing the purpose of the linked data initiative, of a collaborative web [Tudorache (2007)].
- Change management and ontology evolution tracking such that history of the modified data is kept [Liang et al.(2005)]. This is a 'sensible' point as time evolution is a hard subject among ontology developers.

Alongside the Protégé editor, there is an entire API used for backend support for working with RDF/OWL graphs, with the possibility of adding new plug-ins to protégé based on the developers needs.

4.3.7 Limitations

Despite of the elements presented above, the web semantics remains more of a 'research' project, adopted in specific environments, rather than on global scales. There are still several limitations to the current semantic web technologies that leave place for improvement (in comparison with the old relational data base model). We mention some of limitations expressed in the literature with the note that some of these may be in the process of being solved.

Expressivity – OWL DL addresses a fairly small space, the one of Description Logic, in order to preserve the reasoning capabilities. This limits as seen earlier the area of sentences that can be expressed. This is partially resolved via the usage of rules, but the gap remains (e.g. inferencing with OWL that a person with a first and last name has also a full name) [Grau et al.(2008)]. Reasoning must be done with coherent models, and the current web knowledge is full of inconsistencies and incoherence.

Advanced inference – ontology models, in order to be reasoned upon must be treated as a whole (e.g. loaded into memory [Jena (2010)]), as reasoning can't be done unless all the triplets are provided. This is costly for large scale ontologies. If we are to represent the entire web and transform it into an ontology, then inference engines should find ways of solving this problem.

Mathematical operations - the lack of advanced mathematical support is a real problem, when dealing with numerical values and properties [Iannone and Rector (2008)]. With SQL models, the presence of operations such as AVG, COUNT, SUM or numerical computation formulas permit users to transform data in analytical forms (data warehouses). With OWL models, basic SPARQL provides none of these functions. The SPARQL ARQ extension provides support for COUNT and AVG operations but in a restricted format (no combinations are allowed) [Sirin and Parsia (2007)]. On the other hand, as a positive point the usage of string functions (such as regex) is supported with the new SPARQL extensions. Announcements from the W3C acknowledge these misses, and the 2.0 version of SPARQL should have a much better mathematical support.

Query chaining – strongly used with SQL, queries in queries over advanced operations (such as AVG, SUM etc.) are not supported, not allowing the combination of a higher exploration of the triplet space. This restricts the information that can be extracted from the ontology. We consider the following example.

Example. From a Human ontology we want to select all the Father instances that have exactly 3 children. This is not possible explicitly with OWL (even with the ARQ support). We show the differences between an SQL query and an ARQ query that isn't (but MUST be) supported:

Table 3 : SQL vs ARQ query chaining

SQL syntax	**ARQ Potential syntax (not supported)**
SELECT Father FROM DB_Father *(SELECT Father, COUNT(Child) AS cc* *FROM DB_Father GROUP BY Father)* *WHERE cc=3*	*SELECT ?f WHERE* *{* *?f rdf:type :Father.* *LET (?cc := SELECT COUNT(c)* *WHERE* *{* *?f rdf:type :Father.* *} GROUP BY ?f)* *FILTER (?cc = 3).* *}*

In completion with the limitations above, [Rector and Stevens (2008)] present three main key issues with adopting OWL in knowledge driven applications in a large scale environment:

Predictability the question of software reliability with new technologies is always posed by the industry. How reliable are OWL solutions, how rapidly failures can be solved and how debugging can be done with software applications using ontologies is still questionable with RDF/OWL. The main sources of unreliability are:

- Explosion of reasoning and classification times with small changes to the OWL model.
- A single inconsistency error (e.g. declaration an integer value in the place of a double for a datatype property) leads to the fail of the entire knowledge system.

Usability – OWL ontologies are hard to understand as they require a prefect knowledge of the model and the eventual associated rules.

- OWL ontologies are classification depended; reasoned classify ontologies from scratch, thus demanding only occasional connections for the newly inferred knowledge. When working with large scale ontologies, the results are only seen periodically after an entire series of changes. Classification with large ontologies after each modification is not feasible due to the high classification times.
- Hidden models that result from inference. The OWL axioms have global impact. A local control over the results of axioms are very hard and sometimes not possible (cannot classify only a part of the ontology)
- Unlike RDFS, OWL bases its assumptions on what cannot be said, thus the inverse development logic must be adopted for modeling OWL ontologies.

Rich meta-data – many ontologies exist only to index their meta-data, becoming marginal and depended on the annotated ontology. This is contrary to the purpose of ontologies: collaboration. The standardization of the essential information is not always assured. This is actually the standardization problem, where everyone defines his ontology and sets of associated ontologies, which in turn remain in its 'corner'. The 'reinvention of the wheel' is a predominate process when building ontologies, with more duplicates than actual innovation.

4.3.8 Perspectives

Back in 2007, Berners Lee gave three depending directions to the development of information representation via the web. These are: the collaborative web, the semantic web and the web of confidence [Daconta et al.(2003)]:

Figure 29 : Web evolution in the vision of Tim Berners-Lee adapted from [Shadbolt et al.(2006)]

The collaborative web is highly adopted today, with the social networks and internet communities such as Facebook, Yahoo, Youtube or Twitter. Everyone is able to share the information, respond to what others think, get in contact with them etc. The collaborative web is strong, but information remains unorganized.

The next step is the semantic web, the purpose of OWL being to provide a solid and robust platform for a future semantic web global adoption [Grau et al.(2008)]. This will come with:

- More *syntactic* extendibility, such that OWL offers a macro-system allowing everyone to define their custom syntactic shortcuts
- *Querying* – though semantic query, users are able to access exactly the information that they are looking for. Transforming users' demands into the appropriate SPARQL queries (provided the semantic data exists) is a challenging task.
- *Rule Integration* –with all the available rule profiles a decision over a global standard is a must. The RIF fromat seems to be the closest to this. Rules carry with them the issue ontology consistency.
- *Non-monotonic extensions* to enable OWL to solve advanced modeling problems (i.e. exceptions)

The web of confidence still remains a fantasy [Segaran et al.(2009)], but once the semantic web has been adopted, users will then able to concentrate more on confidence and trust.

4.4 Ontologies, information systems and autonomic computing

We present further how semantics was put to use with Knowledge Management in Information System and Autonomic Computing adoption solutions. This represents also the specific foundation of this thesis.

4.4.1 Data modeling vs ontology modeling

Before approaching the two application areas, a comparison between data modeling and ontology modeling is made, starting from [Vrandecic et al.(2010)]. Data models include data bases or XML schemas, usually describing the structure or integrity of data sets. Several points of comparison are identified, as follows:

Genericity – data models apply very well to closed, singular applications. A data model (e.g. data base) will work very well for the specific designed application, and will likely never be used by another application. The notion of 'collaboration' and information reusability is a weakness of data models. Ontology models benefits from the collaborative aspect, as information is shared and data is inter-linked. An ontology model suits the application and the community, increasing reusability and improving the model (as a collaborative effort).

Changes – data model are static. The structure of the model doesn't accept change easily. Reorganizing a data base model is a laborious task, and can lead to unpredicted results. As reinforced by [Maedche et al.(2003)], ontology models are very suitable for changing models due to the fact that knowledge is dynamic. The drawback is that an ontology model is harder to maintain at change level with an increased risk of incoherence.

Operation levels refer to the level of abstraction. For instance, with base models rules are expressed on a low-implementation level (e.g. data type or key restrictions). An ontology model permits the extension of rules towards higher levels (coherence, identity, knowledge deduction etc.)

Expressive power is perhaps the strongest argument in favor of ontology models. Data models include expressiveness of data integrity (data types, key integrity etc.) but provides no support whatsoever for domain conceptualization. Elements such as class definition, property restriction, taxonomy hierarchy are part of ontology models. On the other side, higher expressivity implies higher complexity.

Purpose – inevitably influences the model choice. Depending on user needs, application objectives, client requirements, etc., either of the models can apply. Questions posed are: what is the level of granularity of the information, is there a need for semantic expressiveness, how simple is the domain we want to model. Ontology models even if they are intended for high reusability, in practice prove to be sometimes domains specific, and their usage with other domains (other than intended) generates ambiguities. It is like trying to put a four finger glove on a five finger hand (it can fit but there will be some problems).

Extendibility from this point of view data models lack, as once the model is chosen, extensions are not likely to happen. On the other side, ontology models prove to be easier to be extended (as it is their purpose).

4.4.2 Ontologies and knowledge management - OKMS

The literature contains several cases of using ontologies with knowledge management in Information Systems. In [Maedche et al.(2003)] the authors study the impact of ontologies for managing knowledge with enterprise systems, in what is called an *Ontology-based Knowledge Management System (OKMS).* The OKMS is similar to the classical *Data Base Management System (DBMS)*, where data bases are replaced by ontologies.

The first step with an OKMS is to identify the needs of the enterprise applications and their compatibility with ontology models. The problem being standardization and large scale adoption, enterprises prefer using classical DBMS solutions. The propositions of [Motik et al.(2002)] provides a suitable concept-modeling approach for business wide applications, while pointing out that the two systems (OK and DB) should work together, interconnected and not apart. In order to do this several requirements must be assured, such as:

- *Unambiguous Semantics* - system description should be clear without inconsistencies or ambiguous semantics.
- *Object-Oriented Paradigm* – knowledge modeling should follow the object modeling approach. The object –oriented approach is very intuitive, permits the organization and separation of blocks and enterprise structures, can be easily visualized and is simple to understand.
- *Meta concepts* – Defining hierarchy relations and inter concept/instance relations reinforce the use of meta concepts with enterprise KM. The *modularity* aspect is deduced from meta-concepts, keeping a clear separation of modules and increasing reusability
- *Lexical Information* – one of the key reasons for ontology usage. Expressing the elements of an enterprise system rely on different semantics and meanings. Having a lexical vocabulary ensures that there are no misunderstandings when speaking of these elements.

- *Light-weight inferences* – the ability to deduce pieces of information from the existing one. Enterprise knowledge is not always explicit and depends on the validity of other knowledge. With inference rules, the problematic of scalability and coherency should always be taken into consideration.

Another important aspect that the authors approach is the migration towards the OKMS, as the information already stored in DBMS must be transformed into meta-data and expressed with ontologies. This is a high time consuming process, from where the requirements of the usage of semi-automatic transformation tools. The general architecture of an OKMS is shown in Figure 30:

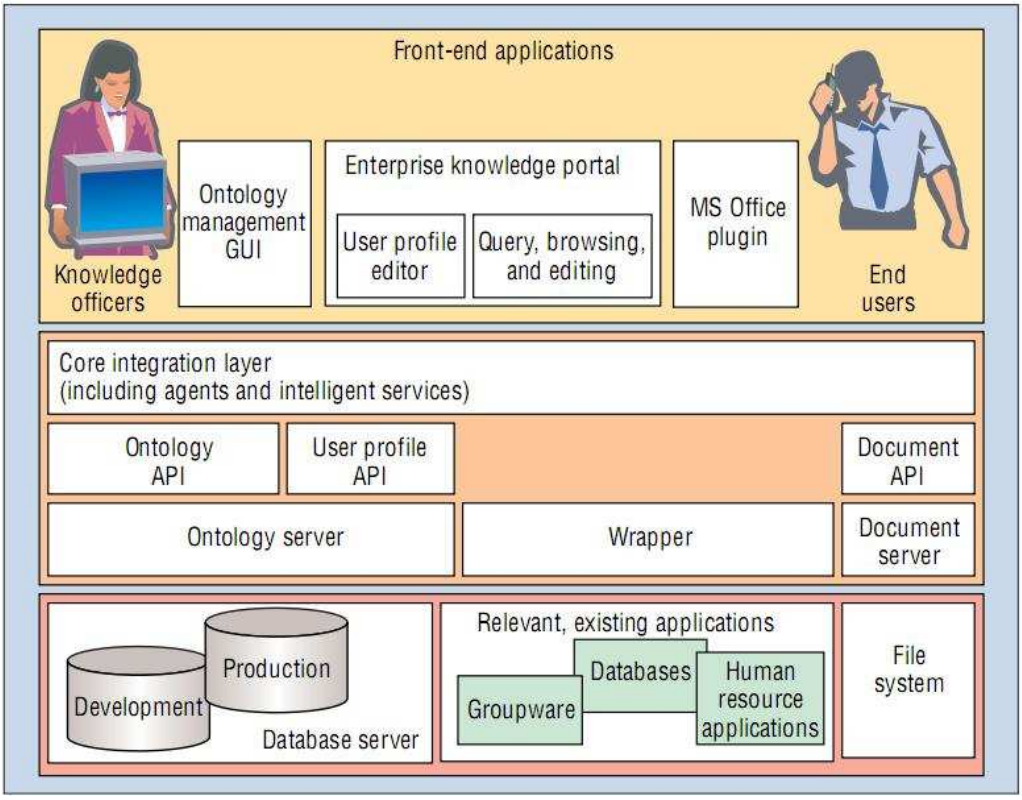


Figure 30 : Architecture of the OKMS [Maedche et al.(2003)]

There is a separation between three layers, somewhat correspondent to a front, middle and back end of a software application.

The front end (on top) provides on one side access to users who profit from the knowledge and on the other side access to the knowledge experts that are in charge of managing the knowledge base.

In the *middle end* (in orange) we find the db and ontology wrapper interfaces, as well as the ontology server and the document server. These contain the information relative to managing the enterprise, in the non structured form (documents) and in the structured from (ontology). The correspondents APIs (document, ontology etc.) are part of the layer.

The backend layer (at the bottom) provides with the existing applications and DBMS system. It contains the information that is to be translated into the ontologies, both manually and automatic (where applicable). The objective is not to replace the DBMS but to make the two work together.

With this architecture, [Maedche et al.(2003)] insist on two processes: multiple ontology handling and the ontology evolution process.

4.4.2.1 Multiple ontology handling

In order to pass from the DMBS to the OKMS, a process of ontology mapping is required from the existing elements and the new concepts (the passage from data to meta-data). Usually an OKMS contains more than just one ontology, the case in large enterprises. Therefore there is a problematic of assuring heterogeneity between multiple ontologies, with a proper interconnection and reusability. The authors identify three approaches for combining ontologies:

Ontology inclusion – which means the inclusion of other ontologies into the working ontology. This is the simplest of the approaches, and creates dependence trees (similar to SDKs libraries inclusion system). There are two drawbacks: (i) the included ontologies can only be included in their 'full' and (ii) the information is 'read-only'. (i) means that for the use a small group of existing concepts, we have to import the entire ontology not only the useful concepts. (ii) refers to the fact that import operation are write 'proof', therefore if a concept is missing we have to create it in the importing ontology, even if it conceptually belongs to the imported ontology.

Ontology mapping is a more complex approach, which takes concepts from the source ontology and converts them through mapping to the destination ontology. This can prove to be more efficient when only the needed concepts are mapped. The problem is scalability and flexibility of with the mapping processes. A change in a base (used recurrently throughout mappings) source ontology can trigger large remapping processes.

Source data mixing is the most complex approach. It implies building a common central ontology from several data sources, each using a completely different ontology; Such operations are very costly, and information retrieval is assured by *mediators*, which have the specific role of formulating queries over the common ontology.

The ontology mapping process consisting of 5 steps: lift and normalization, similarity extraction, semantic mapping, execution and post-processing. *Lift and normalization* helps the first transformation from the lower level of DB and non structured documents to the ontology level providing the first completely structured format. Next, the *similarity extraction* assures the base and derivate vocabulary that will be used. Concepts that are similar, identical or related are specified here. Advanced computations are required to build the similarity matrix. The *semantic mapping* describes the source-destination concept relations, which in turn are part of a mapping ontology (the meta mapping rules). *Execution* is the actual transformation process which loads the destination ontology, and can be either static or virtual. Static execution implies a single transformation from the source to the

destination, and future changes to the source ontology are not visible to the mapped ontology (unless a new execution is specifically made). *Post processing* is the last stage where any eventual post loading operations are performed.

4.4.2.2 Change management

The second aspect of ontologies with enterprise knowledge management is the evolution of knowledge. As [Maedche et al.(2003)] presents it: '*Knowledge Management Systems are not developed to remain stable. Rather, several factors make them subject to continual change'*. In [Zablith (2009)] the author remind the two methods for ontology evolution: the user manual modification and the dynamic, machine learning, evolution. A more detailed view of the ontology evolution is given by [Haase et al.(2005)], focusing on keeping the consistency of the models, after changing has been performed.

An overview for the various phases when updating an ontology, from the knowledge discovery to the validation of the built ontology is shown in Figure 31

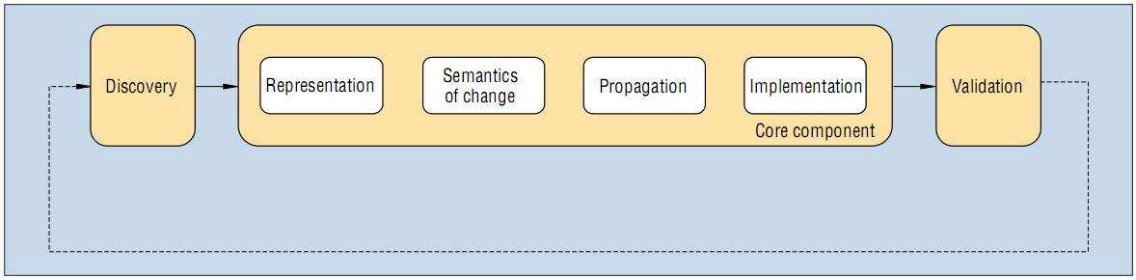


Figure 31 : Stages of ontology evolution [Maedche et al.(2003)]

There are three main stages: discovery, core component and validation.

Discovery refers to the processes of inserting into the ontology new interesting knowledge. This is either a machine based process (e.g. discovery of web services) or human driven (e.g. specific information from readme documents).

The core component includes four sub processes. First we have the *representation* of the discovered knowledge under the form of the existing ontologies. Second, the *semantics of change* refers to observing how the new changed information impacts the existing ontologies, if inconsistencies are created, etc. Third, if the changes are valid, the *propagation* is in charge of updating all the depending ontologies with the new information (i.e. update all the ontologies that include a changed ontology). Last the *implementation* decides which changes will be ultimately accepted with user control. The knowledge expert can either accept or reject changes, reinforcing the fact that the human knowledge expert always has the 'last word'.

Validation – the last stage, once a list of changes has been accepted for an ontology, collaboration occurs and the entire community must agree with the implemented changes.

4.4.3 Ontologies and autonomic computing

We have seen how ontologies can be an information model for knowledge management in IS through the OKMS model. Ontologies have been presented by different authors as one solution for representing the knowledge base with Autonomic Computing adoptions. Nevertheless, we mention that references are not many, thus we are exploring a somewhat 'virgin' domain.

The first step towards using otnologies with AC is done by [Stojanovic et al.(2004)], by presenting a model expressing correlation for a software dependency application. Ontologies are presented as the backbone for correlation engines, and are exploited for their expressiveness and inference power. The authors identify three advantages of using ontologies:

- ***Interoperability*** - ontologies provide a common shared understanding of the domain, both by humans and by machines. AC systems include several managers, which communicate between them and with the humans, and require a common standard knowledge representation.
- ***Machine understanding*** - derives from interoperability, as software is able to process ontology data. This provides autonomic reasoning and several user tasks (even high level tasks) can be performed by the machines instead of the user.
- ***Higher service levels*** - with semantics, the spectrum of provided services (such as verification, justification, configuration etc.) is increased, expressing the four principles of AC, allowing the system to assure itself an improved functioning. Providing new services is actually the argument that the authors use to reinforce the usage of ontologies.

The specific use case is done on correlation engines, as the four states of the ACMs need constant correlation due to the usage of ECA rules. Three types of usage are identified: (i) *data filtering*; ignoring redundant or useless information*;* (ii) *thresholding* , e.g. raising alerts when a measure passes defined limits; (iii) *sequencing* – organizing execution of actions based on their dependencies. Each of these three correspond to an ACM phase (monitor, analyze, plan).

A division into three layers is made in order to express the MAPE-K loop, with presentation of the ontology model: *the resource layer* (monitor), the *event layer* (analyze) and *the rule layer* (plan and execute). This is shown in Figure 32.

A *resource* is a general concept that can fit to anything susceptible for management (physical server, machine, data base etc.). It is pointed out that unless resources share information with the other resources, and provide a global system self awareness, the goal of the AC is not reached. The possibility of organizing resources in hierarchies and of using inheritance between generalizations and specifications is presented as a very powerful argument for using ontologies.

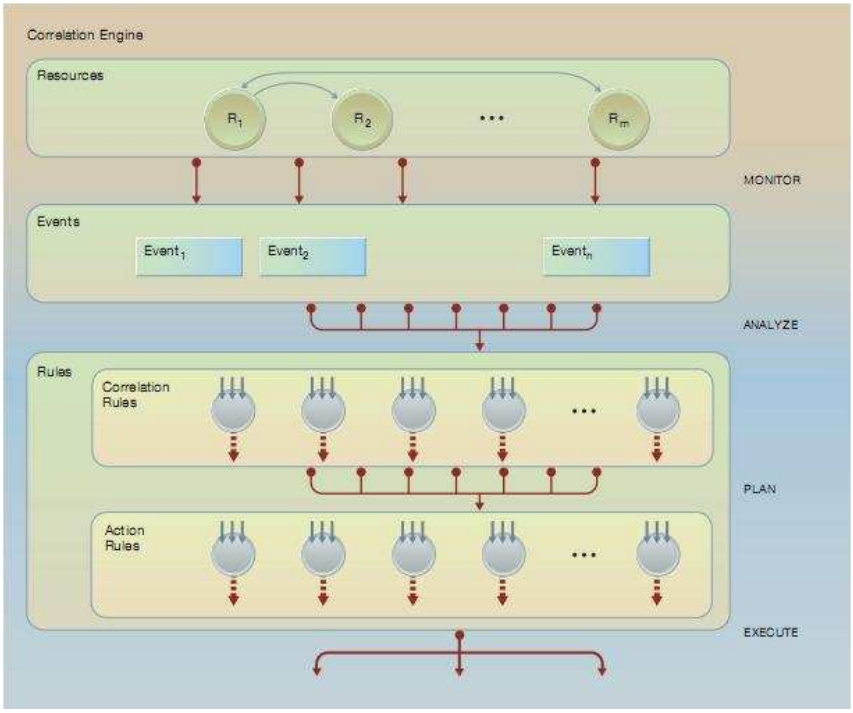


Figure 32 : Layers with the MAPE-K loop phases [Stojanovic et al.(2004)]

Events formalize the changes that occur to resources, most of the times used to identify when something 'bad' happens, assuring diagnostics (from where the analyze phase). Event analysis is not trivial, and requires that the input data is formally organized (not the case with data models). There are two aspects here: (i) formalizing event data with ontologies and (ii) building a meta ontology of events, as events themselves are part of different categories, are interconnected etc.

The rules are dividend into *correlation rules* and *action rules*.

- *Correlation rules* decide weather or not there is a need for action with the diagnostics from the events (planning). Correlation rules usually reduce the number of events that will be triggered, providing a filter mechanism for unnecessary actions or changes. In turn correlation rules can either be *stateless* or *state-based*. *Stateless* rules express punctual actions, such as threshold violation. They are applied at a specific point in time for a static analysis. *State-based* rules are used for analyzing events over time (e.g. the number of times a measure has passed the threshold in the last week and its average value).
- *Action rules* trigger the decided correlation rules actions, usually to resolve problems (e.g. restarting a suite of software programs due to crashes). Action rules replace a part of the human actions by executing autonomic actions and provide with feedback on the new state of the system. In turn, this knowledge is both useful for the human expert and provides an additional base for future ACM loops.

Starting from this idea of combining ontologies and AC, [Gonzales et al.(2007)] showed an integration of the two for device communication within residential homes. Their application was on how to manage new devices with residential environments, such as

PDAs, smart phones, laptops, surveillance cameras, sensors. The approach is based on the OGSi [Alliance (2005)] specifications for the network environment. The architecture is built with the help of intelligent agents that interact with the environment and with the communication devices.

For each new device that is to be integrated with the network, a process of communication between the autonomic agent and the ontology is done, expressed in three sequential steps:

- Agents gather information about the environment, in the monitoring process, and in concordance with the OGSi framework. This information contains elements such as: devices, platform characteristics, installed software etc.
- The agents access the ontology repository and fit the new information to the existing ontology. This is basically a process of instantiating existing classes with the specific environment.
- The agents offer the new information by an unique identifier (URI), so that the service provider can offer the clients the new services (with the highlight on the shared aspect of ontologies)

All of the information, including software, devices, services, the OGSi platform and users profile is modeled with the help of ontologies. The MAPE-K loops make use of SWRL rules to integrate the state passage. The global architecture is shown in Figure 33, with a separation into three layers: resource, events and rules, and the integration of the ACM.

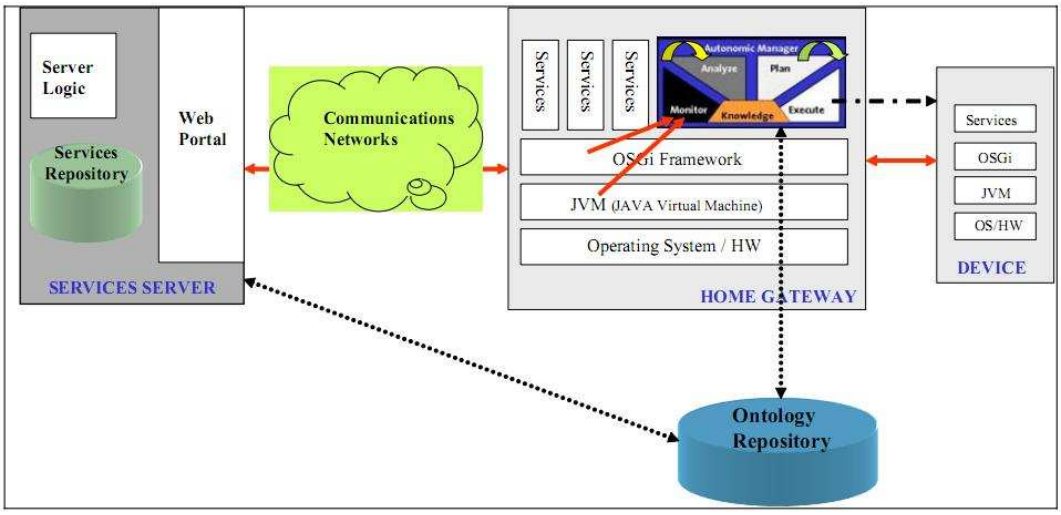


Figure 33 : Ontology and AC adoption application architecture [Gonzales et al.(2007)]

The ontology repository is in link with both the service server and the home central gateway. The ACM is in the middle of this gateway and provides the integration of the new devices through his loops.

Another approach for combining AC with ontologies is detailed by [Normand (2007)], over a study case of network devices failures within large companies. Here, knowledge

engineering at the intersection of Artificial intelligence, Web Semantics and Autonomic Computing systems.

The focused autonomic principle is self-healing, as the presented solution identified the source of failures (network failure, printer no longer working etc.). The topology of the network is modeled with an ontology (very suitable from this point of view), along with all the elements that defined the company (such as users, administrators, departments). The self healing principle is assured by the usage of several autonomic managers (corresponding to the each printer in the network), that implement a specific MAPE-K loop. Then, all these 'small' entities would aggregate to a global ACM (the touchpoint ACM). The architecture model is presented in Figure 34.

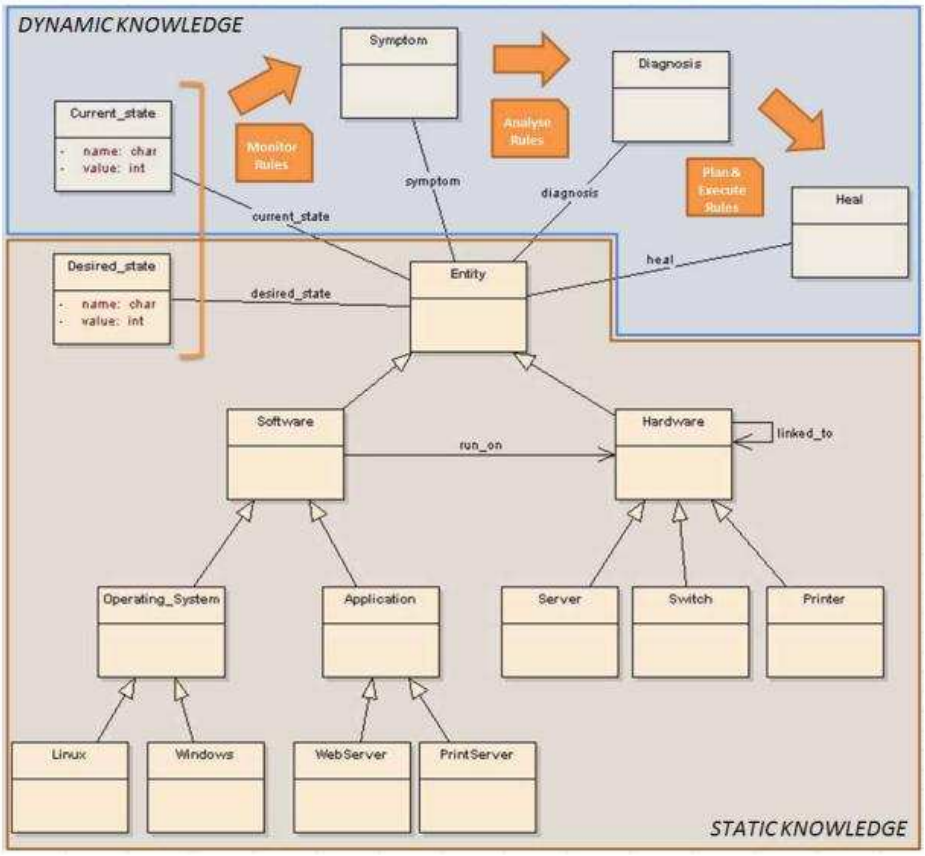


Figure 34 : Autonomic Computing MAPE-K loop global architecture [Normand (2007)]

There is a distinction between the dynamic (the loop passage) and the static (system description) knowledge. The presented solution encapsulates into a single program, reinforcing the idea of a common central control entity. The Jena framework is been used for the implementation, and rules are presented under the form of ECA over the Jena inference rule engine. The presented system detects two types of diagnostics: *simple* (direct ECA analysis) and *complex* (combinations of several diagnostics which lead to more specified diagnostics). For this, the SWAM ontology is created and integrates the presented knowledge base.

There are several general problematic points with the ontology and AC approaches, over which authors in the literature agree. These are:

- Debugging difficulty with the ontology tools; debugging a knowledge base software implies a complete knowledge of the architecture and the elements. This has also been reported by [Stuckenschmidt (2008)]
- Rule system stiffness - a higher number of rules leads to a higher complexity leading in turn to difficult rule management
- Standardization – as several formats and tools are used. For example, there is still no standard for rule interchange. Nevertheless, we can conclude that OWL is an adopted standard for ontology representation.

4.5 Conclusions

The chapter described the main aspects of knowledge management, with a focus on web semantics and ontologies and their application with IS management and autonomic computing. We have seen how the information evolved with the development of the WWW and the main standards for knowledge formalization. Ontologies were presented in the center knowledge formalization, with technologies such as OWL, SWRL or SPARQL. From their description, we understood that they represent a viable choice for knowledge base implementation, and that data base models suffer in areas where ontologies have proven themselves (such as scalability or data model flexibility). We equally have seen that the semantic technologies are in plain development, with the Web 2.0 and that industrial maturity still requires more time for large scale adoptions (such as the Oracle 10g RDF support).

The last part was dedicated to the application of ontologies over the autonomic computing adoption model, as we explored the ways in which semantics enhance ACM knowledge bases and consequently IS management. Surprisingly (or not), there were not many references that combine the two, mainly due to their level of maturity (the final version of AC was elaborated in 2006 where as the 2.0 version of OWL was finalized by the W3C in late 2009). Nevertheless, these existing works prove the interest of the approach and offer many ways to explore, such as the one of decision support and data warehouse management which we describe further in the last chapter of the state of the art and in direct link with our problematic.

5 Decision Support Systems and Data Warehouse Management

"Give me what I say I want so I can tell you what I really want"

The data warehouse expert motto, W.H. Inmon

5.1 Introduction

The last section of the state of the art focuses on a specific type of information systems: Decision Support Systems (DSS). They are presented in opposition with operational systems, with their specific characteristics and way of functioning. Most of the observations relating to autonomic computing and web semantics are retaken and presented in the context of DSS, and how literature sees this combination.

The first subsection presents the general concepts of the DSSs, such as data warehouses or OLAP, with definitions given by various authors and the specifics of the DSS architecture. Then, we concentrate over the core of the DSSs – the Data Warehouses (DW). Following, the management DSS procedures are presented in comparison with the management procedures presented in the first section of the state of the art. Next, we concentrate on measuring performance for data warehouses and on how DSS focus is not on technical performances but rather on the quality of service and user satisfaction. Last, an overview of how AC can be adopted for DSS is shown followed by conclusions including a general view of the combination of AC, Ontologies and DSS.

5.2 DSS concepts

Definition. *Decision Support Systems* are defined as a specified class of computerized systems that supports business and organizational business-making activities, presented as an interactive software to help decision makers access useful information from various sources [Builders (2010)].

DSSs are based on two areas as mentioned by [Shim et al.(2002)]: the theoretical studies of organizational decision making in the late 50s and the technical work carried at MIT in the 60s [Keen and Morton (1978)]. The design of DSS is split into three requirements:

- Advanced complex database management capabilities, with access to both internal and external data
- Powerful modeling capabilities due to a complex system management model
- A simple and intuitive GUI enabling report generation, easy querying, graph and image capabilities etc.

The first explicit DSS specifications date from the early 70s, when [Gorry and Morton (1971)] presented a framework of managing information systems, raising the 'levels' of information processing as shown the following table.

Table 4 : Decision types over information types *[Gorry and Morton (1971)]*

	Strategic Planning	Management Control	Operational Control
Unstructured	E-commerce	Career paths	Grievances
Semi-structured	Forecasting	Budgeting	Assignments
Structured	Dividends	Purchasing	Billing

Three managing activity types are identified by [R.N.Anthony (1965)]: (i) strategic planning concerning overall objectives and goals, (ii) management control as middle goal oriented activity and (iii) operational control as direct specific task control. Moreover, a description of the main decision problems is shown by [Simon (1977)], from programmed (structured, routine) to non-programmed (non structured, hard to express in a standard form). This separation was later replaced with the passage from non-structured to well structured information. According to these criteria, a DSS is defined as a computer system which deals with a problem where there is at least a piece of semi or non structured information. This implies the involvement of both human and machine in the process.

With the evolution of knowledge management, especially with the internet and the web semantics, DSS gained a new perspective, due to environment interactivity and the interconnectivity between enterprises and organizations. In 2001, [Courtney (2001)] proposed a new decision-making paradigm for DSS, whon in Figure 35:

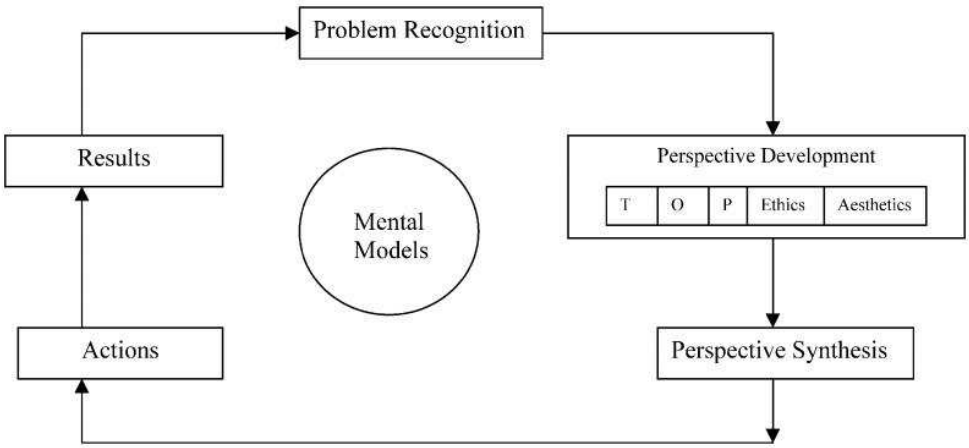


Figure 35 : The decision-making paradigm for DSS [Courtney (2001)]

The proposition implies the development of multiple perspectives (*Perspective Development*), from the point of view of *Technical (T), Organizational (O), Personal (P)* moral *Ethics* and presentation Aesthetics. This is embedded in the mental model, such that recognition of the problem is put in a specified perspective. After the analysis through the

Perspective Synthesis, actions are taken and results can be noticed. Initially, the technical perspective was the main concern for problem resolution, as it was connected to the actual data (database and data models). With modern DSS approaches, the organizational perspective seems to take its fair share, due to knowledge interconnection. Therefore, the tendency for efficient decision making seems to be the same as for the web semantics: collaboration.

On the aspect of collaboration, even 20 years ago, with the customer-supplier interchange over DSS environments, Michael Dell, CEO of Dell Computing said: *"All aspects of the relationship – such as real-time feeds from our manufacturing lines about quality, cost data, product roadmaps, inventory information, and order demand information – are included in the valuechain@dell.com. This allows us to bring our suppliers inside our business and treat them as if they were part of our company. This is an illustration of the virtually integrated business in which suppliers and customers are connected in real time"* [Dell (1989)].

5.2.1 The four DSS pylons

With the development of DSS four main tools are identified as primordial for building the system. These are*: data warehouses, on-line analytical processing (OLAP), data mining and web-based DSS*. A good presentation of the BI and DSS solutions, especially for small and medium sized enterprises is done by the authors of [Grabova et al.(2010)], with details over web based DSS, advanced OLAP (like MOLAP or XOLAP) and data warehousing with in-memory and vector data bases.

5.2.1.1 Data warehouse

The origins of the data warehouse are data base technologies, specifically the relation data model proposed by Codd in the early 80s [Codd (1970)]. Codd spoke of three types of data dependencies: (i) ordering, (ii) indexing and (iii) access path. An ordering dependence expresses a hierarchical relation between data models. The indexing dependence indicates that the index is more than just an optimization technique; it is a link between the application and the data models. The access path dependence expresses how different data sources are accessed by the application, when confronted with changes.

Data warehouses evolved since Codds early model, and were promoted by authors such as Inmon [Inmon (2005)] or Kimball [Kimball (1996)] as a very promising solution for supporting decision making by integration of the operational data. The two authors present the two main different approaches for data warehousing, which will be discussed in more detail in the data warehouse section.

Definition. The definition of the data warehouse, according to Inmon is that it is*: a subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management decisions* [Inmon (2005)]:

- *Subject-oriented* - the data is organized form the point of view of business needs and objectives, rather than from the technical system perspective. This indicates that data warehouses unlike databases are business objective oriented.
- *Integrated* – the data is expressed in a standard format which allows it to be interchanged between various machines
- *Time variant* – indicates that the temporal axe of data is always present, allowing overviews over data history and data changes
- *Nonvolatile* – the way that a data warehouse is build allows data to be either added (insert operations) or queried (retrieve operations). Existing data cannot be modified (update).

Data warehouse opened the door to the development of DSSs and to the next generation DSSs. It is also the core of any decision systems. This is why our focus with this thesis is on data warehouse management. A more detailed state of the art for data warehouses, its characteristics and the different approaches is done in the data warehouse section, after the presentation of the rest of the DSS pylons.

5.2.1.2 On-Line Analytical Processing - OLAP

Introduction of the concept of on-line analytical processing (OLAP) and its standards is shown with the development of the data warehouse concept, as it became an important part of the data warehouse. [Janus and Fouché (2009)] define OLAP as a technique for data aggregation which allows users to dig and drill into transactional data in order to solve business needs. The key elements of the OLAP definition also describe the characteristics of its implementation, as follows:

- *Data aggregation* implies the existence of a hierarchical structure and organization of data (e.g. a data cube with several dimensions), over which mathematical operations such as sums or multiplications can be applied.
- *Dig and drill* refers to the capability of 'navigating' through data both between the same levels of data and, more important, between the various levels of the hierarchies. This is done with simple and intuitive interfaces and under acceptable retrieval times. Moreover, it permits the construction of custom reports further used by the user.
- *Solving business needs* – is what the objective of OLAP is, that is responding to user business problems. The reason that one would navigate through all that aggregated data in all directions is that he searches an explanation to one or more problems which have occurred (e.g. why did women shoes sales drop 50% last month on the western side of France?).

Usually, data throughout enterprises is not stored in a single isolated format. It is spread over several different data sources, most of the times incompatible with each other. One of the objectives of the OLAP technology is to make this data available in a single format (e.g rapport). Compatibility with data means that data from a specific repository should be correctly interpreted and used with the totality of the other repositories. Consider the following compatibility example, adapted from [Mailvaganam (2007b)]

Example. An executive of a sportswear company is interested to find out which are the 5 most popular sport items by customers with the age between 14 and 20 years. Customer ages are stored as birth dates in the purchase web base and as absolute values in the sales data base. A data incompatibility arises between the two, even though they express the same thing.

As mentioned earlier, OLAP is a part of the data warehouse architecture, yet it is not always compulsory to build a data warehouse for OLAP analysis. This is due to the existence of the so called OLTPs (On-line Transaction Processing) which store data from the operational systems. In fact, OLTPs are the same as any other databases, with the main difference being the way in which data is stored. OLTPs are designed for high transactional speed and very quick data access (instantaneously); OLAP is designed for analytical needs, therefore data architecture and organization is more important than data access time, because of which data schemas such as the star design are employed.

Five steps are described for data to be transformed from OLTPs to OLAP and further on to business rapports, derived from the multi-tier data warehouse process layers, as shown in Figure 36:

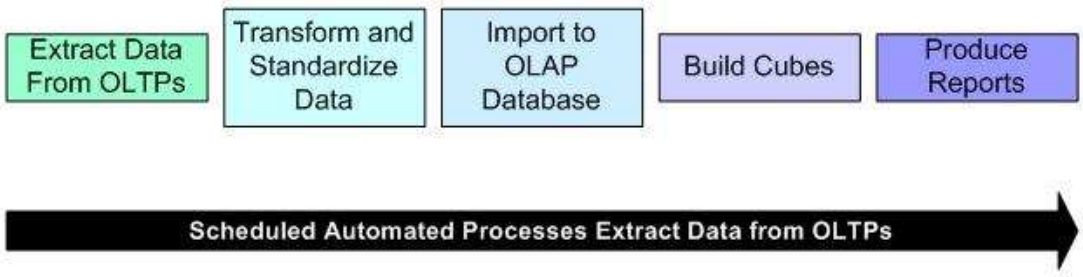


Figure 36 : Data transformation from OLTP to OLAP to business reports [Mailvaganam (2007b)]

First, data is extracted from several transactional bases, and then it is transformed into a standard assuring compatibility. Next, the unified data is imported to the OLAP data bases, which in turn are used to build the multidimensional analytical cubes, which finally are the source for business reports.

5.2.1.3 Data mining

Data mining, as a technique, was introduced to improve and increase the power of OLAP and data warehouse tools, leading to a more complex data analysis. *Data mining* is seen as the analysis of large data sets with the objective of discovering data patterns of interest, such as data evolution [Mailvaganam (2007a)]. Combining data mining techniques with data warehousing is presented as indispensable for business intelligence solutions and DSS.

The objective of data mining with DSS is to: (i) *explain* occurred events, by inspecting the data from the data warehouses or specifically building data reports that would be of interest; (ii) *confirm* - once an explanation is found it can either be verified by other methods

or by an external decision factor; (iii) *explore* which permits the discovery of new information from the existing data, in turn leading to new analysis.

There are two main methods of data mining analysis: *descriptive* and *predictive* [Berson and Smith (1996)], which apply to the data warehouse architecture:

Descriptive analysis, or analysis classification, aims at identifying specific information from the data warehouse, as if trying to find 'the needle in the haystack '.Classification allows the organization of data according to the user needs, different from the data warehouse or OLAP bases architecture. Common techniques for descriptive analysis include: clustering, factorial analysis, hierarchical classification or association.

Predictive analysis is completely different, and focuses on evolution and prediction of data from the data warehouse, based on the already existent data. Data warehouses always implement a time axis. This analysis is more complex, as apart from direct prediction, it also enables chance percentage and probability statics for events to occur. Predictive analysis includes techniques such as: liner regressions, decision trees, neural networks or probabilistic analysis. A good example of predictive analysis with a high degree of probability is meteorological forecast.

The two types of analysis are based on several techniques and tools, from which we remind [Maisons (2006)]:

- *Rule discovery* – refers to two aspects of rules with data mining. The first is the validation of rules that are introduced to the systems (analogy with the consistency rule engines of the semantic web technologies). The second is the deduction of new rules and information starting from the existing one.
- *Decision trees* – as part of the predictive analysis, they enable specific criteria analysis (including probabilistically techniques). Decision trees are usually used to identify the paths that lead to the solution(s) of the problem, and are also able to give the 'best' solution under given circumstances.
- *Signal Processing* – implies the application of various filters to data, such that the retrieved data express specific inter links.
- *Fractals* – based on the mathematical notions of fractals, they are used to find irregular data classification (in opposition to the regular classification techniques).
- *Neural networks* – one of the most advanced techniques of data mining, inspired by the human brain activity, neural networks are used for knowledge discovery and autonomic machine learning. As data pours into the data warehouse, learning methods are embedded with the neural network so that the machine is able to 'understand and predict' the environment and the evolution of the data warehouse.

Business use cases can be designed in perspective to data mining techniques, Figure 37 showing this passage, from the two data mining types of analysis:

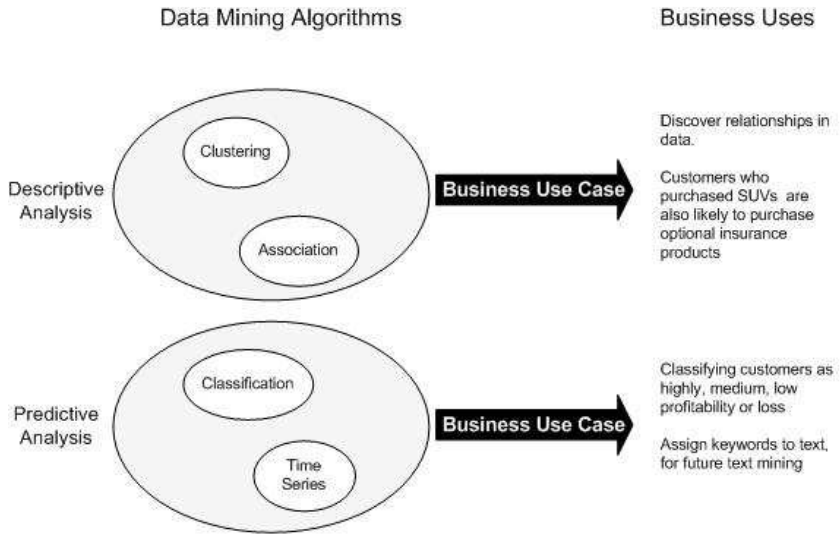


Figure 37 : Building business use cases from data mining algorithms [Mailvaganam (2007a)]

The objective of the data warehouse is to provide an architecture such that experts can focus on the analysis process itself rather than on how to format and integrate data. As Inmon said: *"The data warehouse is integrated so the miner can concentrate on mining rather than cleansing and integrating data"* [Inmon (1996)].

5.2.1.4 Web based DSS

The evolution of web has led in the 21st century to the development of a new 'breed' of DSS: web based DSS. A web based DSS is referred as a computerized system that delivers decision support to the interested user (so far the definition of the DSS) by using web interfaces such as web browsers via the TCP/IP - HTTP protocols [Shim et al.(2002)].

The architecture of a web based DSS is server – client. The server implements a CGI (Common Gateway Interface), that handles Structured Query languages (SQL), post-SQL processing and of HTML formatting. The client can be any interface that allows the user to connect to the web server via the internet (usually a web browser interface). This kind of architecture allows for a more loosely coupled approach of the DSS, and more important allows different users from different locations to enable collaboration with the DSS. Moreover, the development of a web based architecture for the DSS enables a better dissemination of 'best practices' elaboration and adoption, while decisions prove to be more consistent especially with repetitive tasks (e.g. a manager from a company already has taken a similar decision, thus when another manager has to deal with the same situation, he will also rely on the community).

As vendors saw the opportunity of developing web based DSS, a series of tools and products were elaborated to this end. The problem, as always, with collaboration is the lack of standardization and the increased complexity. Each company build its data warehouses internally with little (to be understood none) external relation (i.e. one may adopt a pure Kimball approach; another would make use only of data bases, another of OLAP cubes in an

Inmon architecture etc.). A web based DSS should automatically imply the existence of a standardization and the will of sharing the architecture with the interested parties.

Technical aspects such as charge load and scalability also is an important aspect of web based DSSs. Data warehouses are constructed to be specifically scalable and flexible, to handle huge amounts of information, and even to be susceptible to architectural changes. Over a web architecture, the server should always be available 24/7, handle great numbers of concurrent requests, respond in acceptable times, and cope with the constant growth in data.

The architecture of a web based DSS consists in a three layer design (generally valid for most of the actual web applications), as shown in Figure 38.

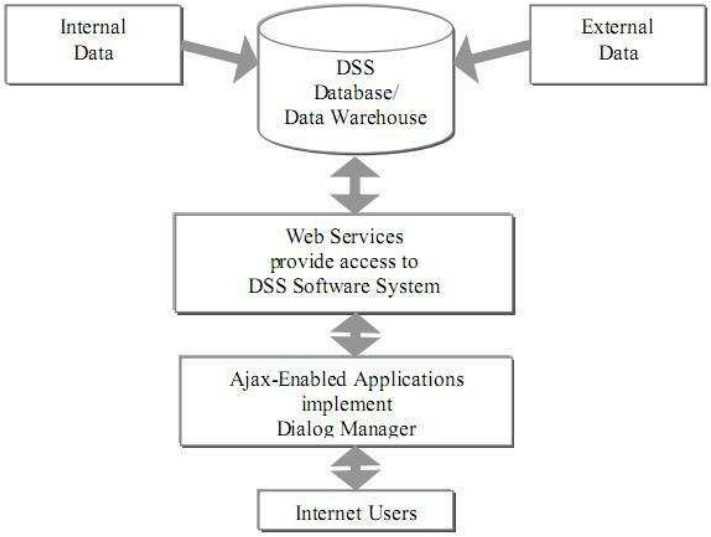


Figure 38 : Web-based DSS architecture [Boreisha and Myronovych (2007)]

- *The data layer* includes the external and internal data as well as the data warehouses that are build with this data
- *Business logic* expresses the best practices, rules and procedures for the business process. It is implemented via web services which provide access to the DSS repositories.
- *Interface layer* – the front end of the DSS, from where users can view reports or eventually modify existing information via the web. Here it is exemplified by ajax-enabled applications which assure the communication between users and the DSS access web services.

What is important to notice is that this is a service-oriented architecture (SOA). The usage of web services with the DSS decision making processes enables a clear separation of components, well defined interfaces, program language and platform independence, scalability, easier deployment. Therefore, the interest towards web based DSSs relies on the promises of global interoperability, the 'dream' of the decision analyst, such that data from a data warehouse is compatible with data from any other data warehouse.

5.2.2 Data warehouses

Data warehouse was presented earlier as the first tool for DSS, influencing the development of the OLAP technologies and the more recent web based DSS. Besides the characteristics and elements that compose the data warehouse, an entire discussion is made in the literature between the two approaches towards its architecture: the Inmon approach based on a well established taxonomical foundation and the Kimball approach for a more pragmatic, on the fly, approach.

A schema of the data flow including the data warehouses is shown in Figure 39.

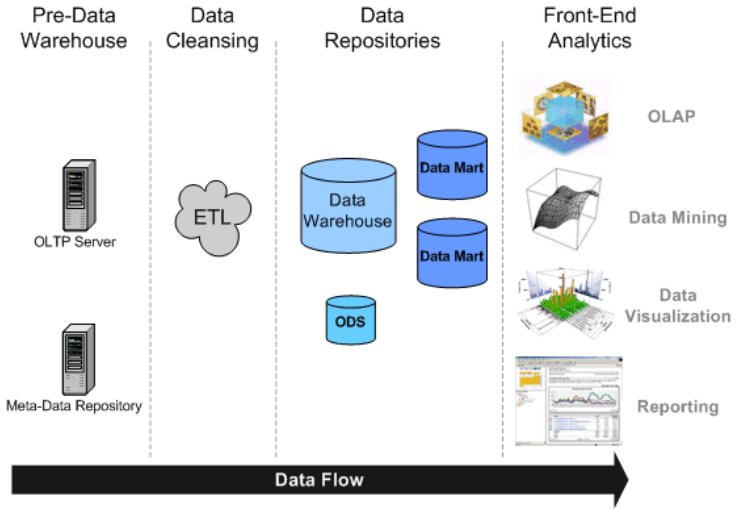


Figure 39 – Data flow chain [Group (2011)]

5.2.2.1 Data warehouse concepts

The data from the data warehouses suits as *analytical data*. If in the early days of DSS operational data was used as analysis support, nowadays a very clear separation between the analytical and operational data is made. This separation is based on several considerations [Inmon (2005)]:

- The data used for operational systems is physically different from the data used for analysis
- The technology used for operational systems is fundamentally different from the technology used to integrate analytical data
- The processing characteristics for operational systems are different from the ones for analytical processing.
- There is a clear separation between the operational and the analytical community

One of the most important criticisms (brought including by himself) brought to Inmon's definition of the data warehouses is the nonvolatile aspect, which draws e huge gap between the operational world. This gap was too large to be ignored, and led to the construction of the *Operational Data Store (ODS)* as a data structure 'nearer' to business [Nelson and Wright (2005)]. An ODS is defined as an integration data base which provides

data for analysis and reporting, similar to the role of the data warehouse itself but for operational systems. Inmon's definition of the ODS derives from the one of the data warehouse with two key differences: (i) specific history of data up to a given limit (e.g. last 3 months) and (ii) a much higher data insertion frequency (e.g. 10 times a day).

As pointed out by [Inmon (2005)] data warehouse is an architecture, a set of rules and practices on how to store and use data for analytical purposes with DSSs. A common confusion is made between this 'meta' aspect of data warehouse and the physical aspect. The physical side of DW includes elements such as: data marts, multi dimensional data bases, data cubes etc. Metaphorically, these are the bricks that build the data warehouse building.

Data marts – a data mart is defined as a repository of data gathered from operational data and other sources with the purpose of serving a particular community of knowledge workers [SAP (2010)]. In the data warehouse context, data marts are the pieces of information contained in the DW, thus extraction and information retrieval from a data mart means extracting a part of the DW. A data mart is usually a purpose specific data repository. A simple example of a data mart is a Microsoft XLS sheet containing various aggregated data from operational data bases. XLS is widely spread among enterprises DSSs.

Multidimensional bases / data cubes are a type of data mart but with a more specific architecture. Unlike traditional bi-dimensional relational data bases, multidimensional bases express data over several dimensions (or axis). The specifications of the multidimensional cube data stores are [Abelló et al.(2003)]:

- Explicit separation of the cube structure and its contents
- Complex dimensions
    - Level structure
    - Instance structure
    - Mathematical formalism for level structure
    - Dimension attributes excluding hierarchies
- Measures and dimension member symmetry
- Complex measures
    - Structured measures
    - Derivate measures
    - Measure operation mixing (e.g. additivity)
- Query formalism
    - Algebra or calculus
    - Ad-hoc queries
    - User defined queries (i.e. aggregates)

A multidimensional data model expression from Oracle is showed in Figure 40:

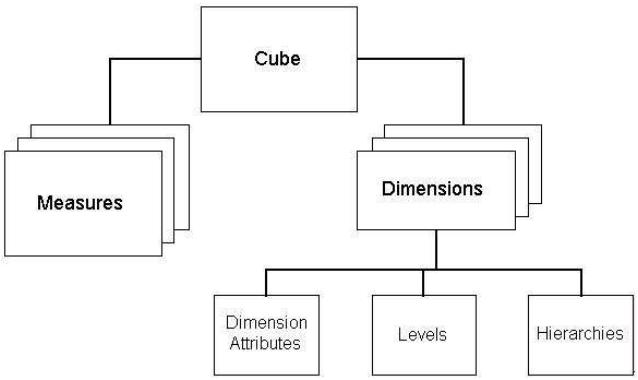


Figure 40 : A multidimensional data model [Oracle (2010)]

At the top of the hierarchy we have the *cube*, a logical expression of the measures over the exact same dimensions. Underneath there are the logical *Measures* and *Dimensions*. Measures have the role of filling the data chunks with the collected data (e.g. price, product etc.). Measures are further on organized by dimensions, a unique set of values that are used to specifically identify data. Usually, dimensions include a time axis, which provides for the evaluative analytical side. In turn, dimensions include three aspects: *Dimension Attributes, Levels and Hierarchies*, somewhat similar to an ontology model. The hierarchies express how aggregation levels are constructed (e.g. Time – Quarter – Month – Week). The level indicates the depth of the hierarchy element in comparison with the first parent (e.g. Time = Level 0, Month = Level 2). Last, logical attributes provide additional information about the data, with various purposes (e.g. display label, last day of the week etc.).

Example. Consider a multidimensional data cube, containing information's about PC sales over each month, in various cities across the world. We notice the intuitive multidimensional perspective, with the big cube. Each dimension intersection crates a smaller cube (chunk of data) that can be retrieved by crossing the respective dimensions (e.g. 5,986 was the average sales for the Executive PC in Tokyo for the two months January and February).

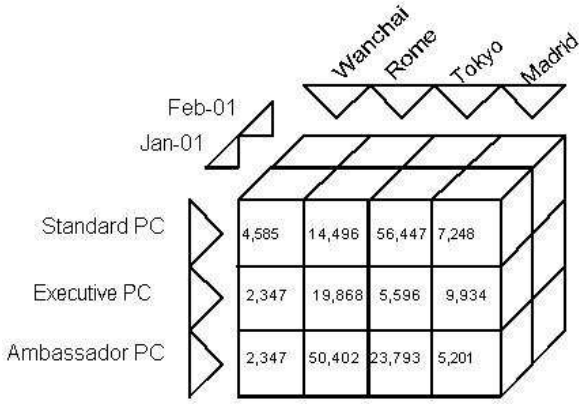


Figure 41 : Multidimensional cube example [Oracle (2010)]

No matter the DW approach or architecture, there are four certain facts to be considered when working with DWs:

- *The methodology* used to build a DW is completely different from the one that builds applications
- DWs are *fundamentally different* from data marts
- *Deliver promise* - meaning that the expectations and assurances given by early DW developers were fulfilled
- DW contain *huge quantities of data*, thus giving a new side to the data management field

The general process of working with data warehouses is described as a multi tier system in which data from one layer is derived from the data from the previous layers. The raw data from the initial sources generate recurrently various derived data. This process however doesn't treat the design of the data warehouse in its fullness; bur rather offers partial solutions to given problems. Problems of data compatibility and interoperability between layers are prone to exist. Figure 42describes the process with five consecutive layers:

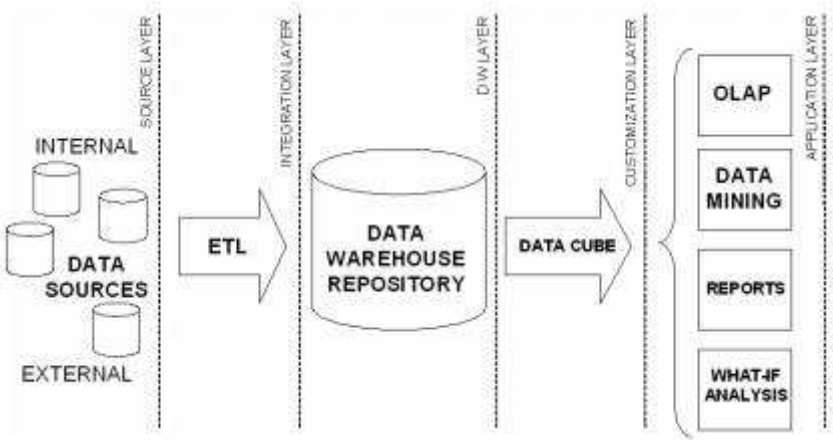


Figure 42 : Multi tier data warehouse layers [Mazon et al.(2005)]

The *Source Layer* contains all the raw data, both internal and external to the system, extracted from data bases, official documents, human knowledge etc. This data, in its current state, is not fully interoperable due to its formatting and expression. The data standardization process is done with the help of the *ETLs*. Once the data is formatted, it is loaded by the ETL into the *Data Warehouse Repositories*. The *Data Cubes* are next build from the DW repository and finally in the last phase, various analysis tools are used to take decisions using the data from the data cubes (reporting, data mining etc.).

5.2.2.2 The data warehouse architecture - Inmon vs Kimball

Literature offers two main visions over the data warehouse architecture, one from Inmon and one from Kimball. Even if they keep in common the general facts about data warehouses, they are fundamentally different. *"…The data warehouse is nothing more than the union of all the data marts…"* said Ralph Kimball back in 1997, whereas Bill Inmon said in 1997 *"You can catch all the minnows in the ocean and stack them together and they still do not make a whale"* [Inmon (1999)].

Ralph Kimball sees the data warehouse as more of a gathering of the data sources; the argument is that the data warehouse architecture, as presented by Inmon, is complex, difficult and expensive to build and requires much thought and investment. The Kimbal design has its roots on the star schema data base structure, with fact tables and a full list of dimension tables that link to these facts [Chaudhuri et al.(2001)]. The granularity between the fact tables is assured with the use of conformed dimensions. A conformed dimension is a unique set of data attributes that are present in several databases from the data model, while keeping the same architecture (structure, attributes, domain values, etc). Data redundancy is somewhat expected with data warehouses, due to the aggregation and drilling side, to provide fast access.

The *Inmon* approach is more towards the architecture and the organization of the data sources (data marts, cubes etc.), with no redundant data, scalable granularity and integrated data design. Thus data marts are focused to human analytical needs, reinforcing the objective orientated DW. Moreover, the Inmon model studies the integration of metadata and unstructured data with data warehouses, along with the lifecycle of the data.

The main differences between the two approaches are presented below, as shown by Inmon himself in [Inmon (2010b)]:

- *Long term vs short term* – if the objective of the data warehouse is speed of development and short term results, then the Kimball architecture is recommended. A long term complex architecture better suits the Inmon model.
- *Scalability* – the Kimball model is a tight model; hardly suitable for major changes in the architecture (similar to the data base models). Once the model has been chosen, any changes to the model imply a series of difficult tasks and can sometimes lead to the entire reconstruction of the data warehouse. With the Inmon approach, there is a better control to this due to the existence of the meta data model. The architecture contains granular, related, relational structures of data.
- *Single truth version* – Kimballs requirements for data duplication indicate that there is no ‘single version of truth’, whereas Inmons model always keeps a single truth information. This is why the latter is slower and more difficult to build, because it brings everything into a big, non redundant, unique picture.
- *Change requirement* – the foundation of the Kimball model isn’t suited when requirements change over time. It is more like an unique period of time for given static specifications. The architectural model of Inmon is more flexible and even encourages change requirement allowing the improvement of the model up to the point where a ‘best suitable’ architecture is found.
- *Concepts* – the components of the Kimball architecture are very few (tables and indexes), keeping simplicity with the data. On the other hand, Inmon’s model is more complex as it also integrates unstructured knowledge, textual data, archive storage and metadata. This enables for a more robust architecture, including the full data lifecycle, the integration of both structured and unstructured information and a more suitable way of a prone-to-changes design.

Seeing the comparison above, it all comes down to the needs of the application, the more complex the more closer is to Inmon's architecture (which is also the architecture used the work presented in this thesis).

5.2.3 DSS and data warehouse characteristics

DSS evolution lead to the changing of DW characteristics over time, translated through the change in the approach of DSS and building the data warehouses. As expected, derived from operational systems, the parallel between the two has always been a comparison point when presenting DW specifics.

From a general point of view, there are five characteristics that identify an informational system as a DSS [Holsapple and Whinston (1996)]:

- Inclusion of a base of interesting knowledge
- The ability to acquire and maintain that knowledge
- Representation of the knowledge in customized ways
- Ability to select subparts of the knowledge for presentation of the user or for the generation of new derived knowledge
- Interaction with the decision makers in a flexible and natural way.

To these five elements, a list of requirements for the high-level DSS is added, as follows:

- Collection of data from multiple different sources (documents, data bases, human knowledge etc.)
- Formalization of data – data collation and formatting, such that the data is represented in a singular standardized form.
- Robust tools for monitoring, reporting and analysis of data
- Well trained DSS experts with the correct skills and expertise

The characteristics presented above are as mentioned general and permit the inclusion of a system in the DSS category. This is useful as to know what a system is designed and used for (i.e. not using the good system for the required job). On the other hand general lines provide further a series of specific properties, which separate the decisional world from its operational sibling. A good starting point the separation shown by Inmon in Figure 43. Inmon shows this list to prove DSS and analytical data require a completely different approach than operational systems. The most interesting differences are discussed below:

DSSs are *subject oriented*, with the goal towards the quality of service and towards the business needs; operational systems are application specific, focusing on application requirements

DSSs represent the *evolution of values over time*, as data warehouses always keep a temporal line. Analytical data expresses the notions of prediction over data, data tendency and evolution. Operational data is accurate, corresponding to a specific moment of access.

PRIMITIVE DATA/OPERATIONAL DATA	DERIVED DATA/DSS DATA
• application oriented	• subject oriented
• detailed	• summarized, otherwise refined
• accurate, as of the moment of access	• represents values over time, snapshots
• serves the clerical community	• serves the managerial community
• can be updated	• is not updated
• run repetitively	• run heuristically
• requirements for processing understood a priori	• requirements for processing not understood *a priori*
• compatible with the SDLC	• completely different life cycle
• performance sensitive	• performance relaxed
• accessed a unit at a time	• accessed a set at a time
• transaction driven	• analysis driven
• control of update a major concern in terms of ownership	• control of update no issue
• high availability	• relaxed availability
• managed in its entirety	• managed by subsets
• nonredundancy	• redundancy is a fact of life
• static structure; variable contents	• flexible structure
• small amount of data used in a process	• large amount of data used in a process
• supports day-to-day operations	• supports managerial needs
• high probability of access	• low, modest probability of access

Figure 43 : Analytical data versus operational data [Inmon (2005)]

Building a DSS and the data warehouses have a different life cycle than the traditional *Software Development Life Cycle (SDLC)* [StylusInc (2008)]. With classical SLDC, the first step is to gather the requirements, then to express them with programs and data sources and then implement and offer the solution to the users. The data warehouse lifecycle implies first the construction of the data warehouse architecture, then the development of the program that uses the data warehouses, and last the description of the requirements. This is due to the fact that when working with data warehouses, one doesn't know exactly what information he wants. The paradigm of the data warehouse user is: *"Give me what I say I want so I can tell you what I really want"* [Inmon et al.(1999)].

Data warehouses are presented as *performance relaxed* as opposed to the operational data, where 'instant' response times are a must. This is one delicate aspect, and will be more detailed in a following subsection about performance measurement. The general idea is that performance measurement with DSS varies strongly, as it is objective and quality oriented, and is not 'stressed' by the real time constraints. Nevertheless, there are limits (e.g. if a manager has his report in a few minutes, it will not consider it as a performance problem, but if it takes an hour to generate it then it might be).

The accessed data with DSSs is in the form of small chunks. 'Mini-cubes' of data are requested by the DSS users as opposed to operational where data accesses are done by units (i.e. line of a table).

The *quantity* of data that is stored and used with the data warehouses is much larger then its operational correspondent, as data warehouses use all the operational data and aggregate it over time. Therefore, working with data warehouses data implies using strategies suited for large data amounts.

The patterns for *resource utilization* (especially hardware) with DSSs are completely different from the ones with operational systems. This is mainly because DSSs are not used constantly all the time. There are periods of maximum charge and periods of complete inactivity, based on the specifications of the system. This discontinuity in utilization makes the prediction of the system usage harder, as it first needs to learn the utilization patterns. Figure 44 shows this behavior:

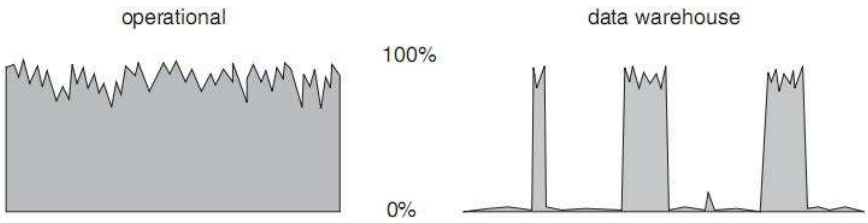


Figure 44 : Hardware utilization patterns for operational and data warehouse [Inmon (2005)]

The operational system is constantly used, with constant data transactions and retrieval, having a full resource charge at all times. On the other hand, data warehouse is only used during the utilization periods, or the critical periods. During these periods resources should be allocated to the data warehouse, such that performances are at their best, where as during inactivity periods, resources should be kept at the minimum required. This constant resource balancing is more detailed in the management procedures, as it is one of the missing points with data warehouse systems today and a solid argument for not mixing the two environments on the same machines.

5.3 Management procedures

Management procedures and best practices with DSS are elaborated with regards to the DSS characteristics presented previously. It is clear that some of the standard procedures for IS don't apply for DSS (e.g. data access management), while are perfectly valid (e.g. system monitoring).

The first important question that must be answered when speaking of DSS management is: who is the DSS user? [Inmon (2005)] describes the user as the DSS analyst, more of a business person rather than a technician (unlike operational systems). His primary job is to define and discover the knowledge in the decision making process, and his objectives constantly change. The DSS user is a researcher, always asking for more data, based on the data that he has. He states that: *"Ah! Now I see what the possibilities are, I can tell you what I really want to see. But until I know what the possibilities are I cannot describe to you what I want"*. This way of thinking is (i) legitimate, as this is how an analyst sees them, (ii) global, as all DSS users think the same and (iii) impacts directly how data warehouses and applications are developed, as opposed to the traditional SDLC.

Once the data warehouse is build, the first step of management is the awareness towards its state, explicitly: monitoring of its environment. *You can't manage what you don't measure* [Globerson et al.(1991)]. Inmon presents a checklist that any DSS analyst

should be able to fully comply with in order to sustain a good management (or at least that the management doesn't get out of hand):

- Growth management, when, where and why data growth is occurring
- Identification of which data from the data warehouse is used
- Calculating the response times on the users reports
- Knowing who is using which data from the data warehouse, and how much of this data is used in comparison with the users that are using it
- Knowing exactly when the data warehouse is in its critical periods and in its non-utilization periods
- Calculating the level of usage of the data warehouse, in rapport with how much data is used and how often it is requested.

As data warehouses are in constant growth, their management must follow the performance guidelines. Response times in DSS are seen as time relaxed, as opposed to OLTP. Yet, time relaxed doesn't mean not important. One may wait for a report to be generated in 10 minutes but maybe has not willing to wait for it for 2 days. The simple fact of considering only the technical response time without integrating the quality of service is inexact for DSS performance measurement.

There is a clear separation between two main utilization purposes that a data warehouse has, based on the life cycle of the data. These two utilization purposes are: (i) active day usage and (ii) night usage [Chaudhuri and Dayal (1997)]:

- *Day usage* – represents the period in which DWs are used for querying and report generation. A high level of user activity is shown during these periods and the response time is critical, as to suit the user expectancies. During day usage, the data from the data warehouse is up to date until the previous day (operational data form the current day is not integrated).
- *Night usage* – or also known as the batch period, in which the data warehouses integrate the operational data from the day that has just passed. These operations usually are very resource and time costly, as cube reconstructions and recalculations are being executed. During this period there is virtually no user activity on the data warehouse, as any rapport generated would be inconsistent. From which the need that during night, resources should be allocated in a manner that the reconstruction operations are over until the morning (the start of the next operational day)

The separation of these two periods implies the adoption of different management policies during day and during night. Unfortunately, the real implementations of data warehouse management don't in most of the cases take into account this separation as our studies at SP2 show. One-time resource configuration (at the deployment of the application) is a common practice, and changes to it occur only when problems have already impacted the functioning of the system.

5.4 Performance measurement

Performance measurement with DSSs brings into discussion, along the technical aspect (from operational systems) the user satisfaction aspect (form the business objective oriented point of view of analytical systems). There are tow main types of performance [Agarwala et al.(2006)]:

- *Raw technical performance* – related to the technical indicators for measuring system performance (such as response times for recovering data from the data warehouse, for report generation, data calculation etc.)
- *Objective performances* – relating to user expectations. These measures are usually based on predefined policies and rules that indicate weather or not satisfactory levels are achieved by the system. (e.g. response time in rapport with the importance of the data warehouse or with the user activity).

One of the most important measure indicators when speaking of DW performance is the *QoS (Quality of Service).*

Definition. The QoS is defined by the levels of satisfaction from the users in rapport with their expectances, and usually is computed as the rapport between the actual value of the metric and the objective defined value of the metric. The exemplification of the QoS computation with regards to two involved factors, priority and level of usage, is shown for a data transmission environment by [Agarwala et al.(2006)]. It leads to a definition of the QoS as shown below:

$$QoS = \frac{Monitored\ Value}{Specified\ Value}$$

and

$$QoS_{global} = \frac{\sum_1^n QoS_i * Scaling\ Factor}{n}$$

The basic single parameter QoS is computed by the rapport between the monitored values and the specified expected values. If several parameters are in charge of offering a global QoS, then a scaling factor is multiplied with each individual QoS, and in the end an average of all the scaled QoS is computed to obtain an overall system QoS value. As an example, E.F. Codd elaborated a series of practices for OLAP bases, which translate to a general well known rule (by DSS experts) that 80% of the OLAP queries should have a response time under a second [Codd et al.(1993)].

The data warehouse quality model is goal oriented and roots from the goal-question-metric model. Whether metrics satisfy or not the goal determinates the quality of the DW [Vassiliadis et al.(1999)]. Data warehouses must provide high QoS, translated into features such as coherency, accessibility and performance, by building a quality meta-model and establishing goal metrics through quality questions. The main advantage of a quality driven model is the increase in the service levels offered to the user and implicitly increase in the

user satisfaction. QoS specifications are usually described by what is known as *Service Level Agreements (SLAs) and/or Service Level Objectives (SLO).*

Definition. SLAs (or Documents of Understanding) are contracts between the consumer and the supplier, indicating the business process, supported services, service parameters, acceptable/unacceptable levels, liabilities or actions to be taken under various circumstances.

They can equally be made between the IT department (as a service provider) and the other departments (as clients), including 'punishment' and 'reward' clauses relative to degree at which SLAs are met [Lewis and Ray (1999)]. The reward may be missing, but SLAs contract violations are well established. SLAs in general are clients oriented, with specification of business objectives over technical objectives. SLAs can cover aspects such as: problem and error fault, configuration, accounting, security management etc. SLOs derivate from SLAs, and concern the more technical aspects.

With DSSs and data warehouses, one important area where service levels are crucial is the utilization periods, in direct link with the answer to the question: what is used, by whom and when? Data warehouse high charge periods are specified with SLAs. When building a new DSS and putting in place the data warehouse architecture, the utilization periods are specified for the various applications. For instance, based on application purpose we can have: the budget application which is used between the 1st and 5th of each month, the sales applications which is used in the last week of each month, the marketing application 2 months / year during the reduction periods etc. Definition of such SLAs permits the determination data warehouse importance and priority at given times.

The key to enterprise performance management is to identify an organization's value drivers, focus on them, and align the organization to drive results [Oracle (2008)]. This is the most simple to understand and yet hardest thing to implement, as Oracle specifies a series of six management processes and the associated key metrics. These are: (i) gain to sustain, (ii) investigate to invest, (iii) design to decide, (iv) plan to act, (v) analyze to adjust and (vi) record to report. To exemplify, the design to decide performance indicators are presented in the table below:

Table 5 : Design to Decide Performance Indicators [Oracle (2008)]

Perspective	**Business results**	**Business drivers**
Financial	Economic added value (EVA) Shareholder value	Customer perspective metrics Process perspective metrics
Customer	Customer satisfaction Revenue growth Brand value	Process perspective metrics
Process	Productivity, operational excellence benchmark (time to market, cost, quality) Costumer profitability Revenue from products	Growth/learning perspective metrics
Growth/learning	Skills/competences match IT effectiveness benchmark Impact (monetary, time, quality) of improvement initiatives	Economic added value

There are two types of metrics, one for the result of business (how to measure the output) and the other for what drives the business. For example, from the customer point of view the most important performance indicators are the satisfaction, the growth of the revenue and the value of the brand, all of these driven from a process perspective.

5.5 Conclusion

In this section we have presented and understood decision support systems and data warehouses, over the main references and characteristics. The natural question that is posed is whether or not technologies such as autonomic computing fit with this specific type of systems.

There is not much reference in the literature to management of DSSs with autonomic computing, maybe due to their 'subjective' nature. Most of the works is done with data bases and DBMS, as they easier to express with objective technical tasks. Technical is the keyword, as it is much easier to express technical needs and render them autonomic than analytical decisional operations. In [Elnaffar et al.(2003)] and [Lightstone et al.(2003)], both authors focus on the data base aspects, but also mention the analytical side of autonomic behaviors.

More specific, a list of several shortcomings is shown by [Elnaffar et al.(2003)] when it comes to analysis, and which are presented below as a parallel between DBMS and DSS management:

- *High human input and intelligence* – translated by the lack of introduction of human experience with the management process; an autonomic decision process lacks the implementation of the humane expertise, by main cause of standardization and information formalism.
- *Dynamic adaption* – perfectly valid for the operational world, especially by real time tuning advisors. The non-continuous aspects of DSS relax this shortcoming, but with regards that data warehouses data change nevertheless (even if more seldom than operational data).
- *Lack of analytical capabilities* – complex analytical models, rules and advanced prediction systems miss form the AC implementations. This means that at least one of the base objectives of DSS is not met by the current AC implementations.
- *Lack of Prevention* – derived from the lack of analytical capabilities, any 'healing' operation will be executed after the problem occurs. Prevention is implemented at best at a base level, and complex prediction over possible problematic behaviors is missing.
- *Standardization* – interfacing an autonomic DSS with other systems proves to be a hard task. On this subject several recent works have been made especially with the development of web based DSS which allowed a loosely coupled architecture and the passage through web services.

From the definitions and applications areas of AC it is clear that its objective is towards operational systems (at least for now). The focus over technical parameter monitoring and improvement or the 'real-time' continuous nature of an AC implementation reinforces this

trend. Analytical objectives are hard to express and implement, and AC looks especially towards the simple tasks (the low level management tasks). Adopting AC to DSS is a dream objective for any DSS expert and would provide a milestone for the future of DSSs and data warehouse management.

6 Solutions and Problematic

"Removing all the possible causes of errors doesn't automatically deliver success"

Anonymous

6.1 Introduction

This section introduces the context of the thesis, with regards to the industrial environment (the SP2 Solutions company), the research laboratory (LINA-COD), and to the problematic, needs and objectives raised by the two worlds (industry and academics).

The chapter first offers an overview of the SP2 Solutions company and of its current offered solutions. Then we show a list of industrial issues in relation with the elements shown in the state of the art. Third, we present the cvasi-equivalence of the industrial issues in terms of academics points of research and problematic. We conclude by mentioning the main objectives of this thesis, in order to solve (or at least try) the mentioned problematic.

6.2 Industrial climate – SP2 Solutions

6.2.1 Presentation

SP2 Solutions has claimed its place as the first software editor specialized in the monitoring and optimization of DSS. Thanks to a constant innovation effort with the main objective of improving the service provided to the DSS users, SP2 places itself as an decisional actor who helps enterprises with their audit decisions. With a team of excellent BI experts and R&D engineers, SP2 offers several services, starting from the proposed software suite: BI Copilot [SP2Solutions (2010)]:

- *Technical Assistance* – assistance with the installation and maintenance of BI solutions such as BI Copilot.
- *Externalization* – expresses the needs of companies to use external service providers to manage their DSS, because internally the operation is too costly or even impossible to assure, due to human resources. SP2 offers to analyze the DSS and offer regular or on-demand rapports of audit and future previsions.
- *Advice* – a service offered to come to the help of companies, by analyzing the information gathered and offering access to a network of BI experts, in accordance to the company's situation and needs.
- *Training* – assures training over the usage of various BI solutions, both for regular BI user and for the technical BI staff of companies (demands of competence gain and advanced detailed expertise).
- *Integration* – capability of the proposed solution to be partially or fully integrated with the DSS of the customer

6.2.2 Offered solutions

The formalized solution proposed by SP2 is a suite of software products, called BI Copilot, with the objective of improving the management of existing decision support systems. There are six benefices and innovative points with BI Copilot:

- *Unified vision of the DSS* – all elements that concern the DSS management are gathered under the same interface and the data is consolidated and stored to a unified CMDB. This enables increased visibility with the decisional system, decreases the time spent by BI experts with their tasks and enables cross analysis.
- *Decision oriented* – the solution is thought with consideration to the characteristics of DSSs. It contains connectors linking with the principal BI software, integrating a specter of responsibilities such as: technique and functional, support and user; and takes into consideration DSS management constraints with its usage periods.
- *Continuous improvement* – assured by integrating user feedback over the pertinence of produced results. It also offers real-time visibility over the efficiency of the DSS. The main benefits here come from the capability of the system to self-improve over time.
- *Forecasts* – enabling future evolutions of the DSS, over its health and performance, by archiving all past information over long periods of time (years) and computing future evolutions by using prediction algorithms. This leads to a decrease of the of the number of incidents (by preventive maintenance) and a decrease in the costs of maintenance (by automatically raising alerts with the potential problems)
- *Full accessibility* **–** the solution is targeted at all BI user categories, such as: technical staff, regular customers/users, directors, project managers, administrators etc. This assures a common language between them, therefore reducing the 'miss-understandings' with human interaction from different areas.
- *Fast and simple* **–** BI Copilot enables several use modes: on demand, integrated with the customers system or on the site. This avoids modeling a full IT system, decreases the costs of maintenance and enables the solution as risk-free.

The BI Copilot architecture contains several modules, each with its own specificities and purpose. This is shown in Figure 45, which distinguishes five modules, on top of a raw data layer, all interconnected with third party software for data access.

The Raw Data layer – it is represented by all the BI applications and software installed on machines and the corresponding data they provide. This includes configuration logs, OS resource usage statistics, software activity logs, warnings and exception traces etc.

(Module 1) Data Collector – the first module of BI Copilot, which gathers the information from the software layer via connectors to the software from the raw data layer.

(Module 2) Operational CMDB – this module is in charge of organizing the data collected by the data collector into data bases (SQL bases in this case), by dividing the information into: configuration data, performance data and exception/error data.

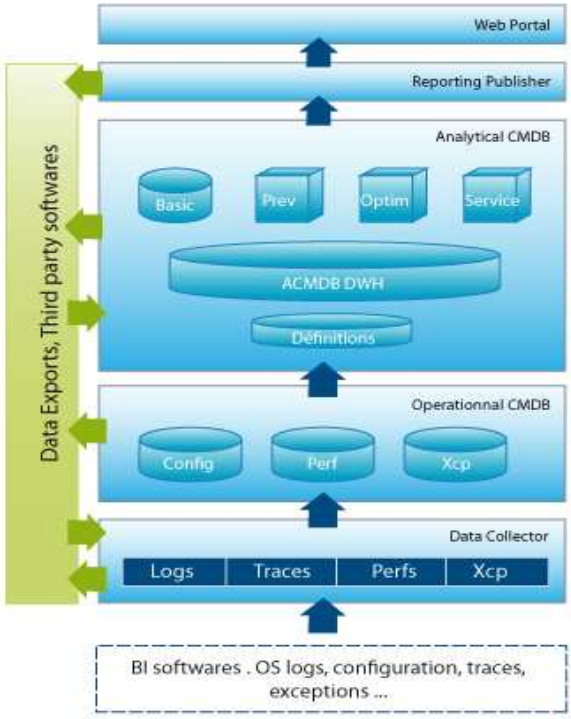


Figure 45 : BI Copilot Architecture [SP2Solutions (2010)]

(Module 3) Analytical CMDB – takes the organized data from the operational CMDB and builds data warehouses based on this data, with operations such as aggregation, crossed calculations between multi dimensions etc. This data architecture is build using analytical forms of storing data, such as OLAP bases or multidimensional cubes. On the analytical data, various operations of analysis are performed for driving the decision process. These include: (i) basic operations such as aggregation, (ii) prediction operations for future evaluations, (iii) optimization operations to improve the system functioning and (iv) service oriented operations to improve the levels of service and user satisfaction. It is also in this module that the works from this thesis are integrated, via the BI Self-X project.

(Module 4) Reporting publisher – this module is in charge of construction of user reports based on the analytical data. These reports reflect any customer-interesting operation from the analytical CMDB such as: prediction reports, warning and failure reports, configuration change reports, evolution reports etc.

(Module 5) Web Portal – the last module at the top of the BI Copilot architecture represents the GUI interface with all the information from below. Users can access the generated reports, view the state of their DSS, the configuration and organization of their DSS; they can equally choose which reports they are interested in and the corresponding commercial solution (i.e. the service offered is pay per each rapport).

Third party software, data export – all the five modules provide the capability of exporting their data and making it available at any time. This way, adoption of the BI Copilot architectures stops with the client needs at any of the modules. For instance, a

customer may choose to stop at the Analytical CMDB module, as he already has the reporting and interface capabilities.

6.3 Present work

The present work of this thesis is formalized in the heart of SP2 Solutions as the *BI Self-X project*. BI Self-X represents the fruit of the SP2's research efforts and is part of the Analytical CMDB module of the BI Copilot architecture. The name itself indicates the main focus of the thesis, which is improved management of data warehouses and DSSs.

In our vision, improvement of the DSS includes several aspects related to a good DSS management, such as: gathering the existing data, integration of the data, intelligent analysis of this data, appropriate prediction algorithms, intuitive interfaces for accessing the data etc. The final purpose is decreasing the costs while increasing the user satisfaction.

6.3.1 Problematic

There are several issues that the works in this thesis tries to approach or solve. We note nevertheless that some of them remain very little or partially approached, whereas others are more detailed with the proposed solutions. To a better separation and understating of the problematic, we divide it into two main categories: Industrial and Academic. We only make at most short references to the solutions we have proposed, as the detailed propositions are presented in the next chapter.

6.3.1.1 Industrial

The industrial problematic contains the point of view of enterprises and companies, reflected by the practical problems they encounter, failure reports, bad management, unsatisfied customers, low service etc. There are three main industrial issues approached:

- *Knowledge management* – the information required to manage a DSS is stored under several formats (editor documents, technical forums, human experience etc.) leading to a laborious and time consuming work when trying to find and use a piece of information.
- *Manual Low Task management* - low level tasks, such as parameter reconfiguration, are done manually, and require most of decision experts' time (over 70% [IBM (2005b)]).
- *Performance measurement* – when measuring the performance levels of a DSS, considerations such as Level Service Agreements (SLA) are not taken into account. Improving a technical performance measure doesn't guarantee the improvement of the quality of the provided service.

6.3.1.1.1 Knowledge management

The problematic of knowledge management refers to managing all information that is used for system deployment, configuration, maintenance and optimization. The efficiency of

knowledge management depends on two main factors: (i) *the amount and complexity* of the information available and (ii) *the formalization* of this information.

The amount and complexity of information – derivates from the principal problematic of information growth in size and complexity with IS. First, this affects the efficacy of the DSS users (both experts and decision makers) as they need to process more elements before acting. Second, it impacts on the financial costs of the enterprise, as they need more qualified and an increased number of DSS experts to manage their decisional systems (and informational systems in general). Consider the following example, taken from the 'failure' experience of a DSS expert.

Example. One of the companies for which the expert was working for, overseeing the good functioning of the DSS system decides to perform a migration change from Essbase v9 to the latest v11. The expert has the knowledge of the two versions, makes an assessment of the situation and performs the migration. During the first month everything runs smoothly, without incidents. At the end of the first month, a user of the DSS requires a functionality of integrating some of the old reports with the new version. When attempting the operation a fatal error occurs, and the import operation fails all the time. One of the release readme documents of Essbase v11 specifies that this operation is known to create a failure and is documented as a known bug. The DSS expert, when he performed the migration, had knowledge over the v11. Yet, he was not aware of this particular issue. Obviously, it is impossible for a human mind to know and take into account all the documented particularities and possible issues for each software version. To resolve this, either a complete rollback should have been made or they could wait for a patch of v11 which would fix this issue. This situation happened with one of our clients, and fortunately we were able to develop a way of 'overturning' the import / export operations.

The conclusion of the example is that knowledge management is vital, and any kind of knowledge dependent operation is prone to human error. Therefore, the tendency towards autonomic integration of the knowledge is a main problematic for DSSs with enterprises.

The formalization of information – represents the second major problem with knowledge management. Mainly, as the amount of information increases so does the various forms of knowledge representation. Nowadays, information is found in plenty of local or public sources (e.g. editor readme documents, technical forums, editor web sites, blogs, personal web pages etc.). And the most important of all is human experience which is the most difficult to formalize. Therefore, enterprises 'fight' for the best DSS experts, which in turn desperately require ways of formalizing a maximum of information from a maximum of sources. Their problem is how to bring together all this information so they can exploit it at its full potential. To better understand how the information formalization is vital for a company, consider the following example.

Example. The DSS expert of a company decides to no longer provide its services to the company, as he found a better offer with another firm. Usually, throughout companies today the number of decisional experts is very limited and they are very rare as a human resource. Once an expert leaves, it takes with him all his experience and his knowledge over the DSS of the company he's just left. So, as the company is forced to hire someone else, this new

DSS expert is put in front of a system he has never seen, with particularities that he cannot yet foresee. This provides a high risk of performance degradation or even failure of the DSS, until the new expert re-discovers and integrates the information and knowledge for the company's system. On the other hand, if the company would have implemented a decisional knowledge formalization procedure, the old expert would have left behind more than a 'closed' system, but also a part of his experience for future usage. This is taken again from our experience, as most of our clients rely on one or maximum two experts for managing their decisional systems.

The conclusion of this example is that enterprises see a problematic in knowledge formalization, and they are more and more aware of its criticality with its growth.

6.3.1.1.2 Manual task management

In the field of information systems manual task management has become a problem with the increase in the number and complexity of the required tasks. As tasks require more and more time and qualified experts to handle, to costs of performing them increases accordingly. Moreover, as tasks are performed by humans, increase in complexity implies increase in the risk that a task is affected to human error. There are tow main issues: (i) *low-level tasks management* and *(ii) manual task performing*.

Low-level task - refers to a repetitive task that requires specific technical notions. Low-level should not be mistaken with easy. But, with the specific technical knowledge, the task becomes easy as no additional information will be needed to perform it. Because of this, the task can be as well performed by a machine, eliminating the costs of the human resource and the risks of human error. The human expert will not be replaced, but will only supervise the execution of the low level tasks. Let's consider the following example.

Example. A DSS expert is in charge of the DSS of a company. His responsibilities vary from configuring the data warehouses to the audit analysis. One of the low level tasks he must perform is to configure the cache allocations of the OLAP bases of the DSS on each server, on specific bases. It is always the same with little variations (same DSS, same OLAP bases) and takes a lot of the experts' time (2 days / week). As the experts contract ties him with the company 3 days / weak, he only has 1 day left to analyze the reports and perform the auditing tasks. An auditing task is considered as high-level, because the data changes all the time and the conclusions on the reports require a human expertise. Moreover, the high level task depends on the low level one, as the reporting should be done with a good configuration and performance. As 1 day is not enough for this activity, the expert is forced to work extra time, stay more to perform the tasks, perform certain tasks in the detriment of others etc. This happens often with our clients, as we, as decisional experts, are always faced with deciding what to do first.

The point of the example was to show that an analysis of the tasks to be performed should always be done before executing the tasks, in the sense of dividing them into low level and high level. Once this division is done, we can refer to the second key point: manual task.

Manual task – a manual task is any task performed by a human, in our case the human expert. Both low and high level tasks are most of the time manual. The problematic of manual tasks is how to render them autonomic under human supervision. We exemplify below, based on the example above, as the two aspects (low task and manual task) are strongly related.

Example. A company calls its DSS expert on the field once each two weeks, in order for him to reconfigure the OLAP bases. The operations that the DSS expert must perform are based on a set of well specified rules that he has knowledge of, and each time he does the same repetitive actions. He identifies these configuration tasks as low level, and realizes that they can be performed faster and more efficient with the help of a machine. Therefore, the task is a strong candidate to be rendered autonomous.

The example above points out a specific type of autonomic low task: configuration and maintenance. Yet, the area of possible autonomic tasks can be extended to include even complicated high level tasks. An example would be system diagnosis or evolution prediction based on the existing symptoms. With our models we aim also at treating some of these higher level tasks, of course, along with the low level ones.

6.3.1.1.3 Performance measurement

The problematic of performance measurement has deep roots with information systems, as this subject reunites two worlds: the world of customers and the world of technicians.

World of customers - As seen from the state of the art the industry searches to improve the service offered to customers; therefore business processes are guided by the quality of the provided service. This is the world of customers, following the 'our client, our master' rule. Measuring performance from this pint of view brings a very subjective side to the problem. Sometimes, quality specifications are very clear and presented under a structured form (e.g. a data base with thresholds for various types of parameters and the related alerts). In other situations, the expectancies are more non-structured (i.e written in the contract of agreements). Consider for instance the following non-structured extract from a SLA: *the application must not fail during its production cycle*. This can mean several things: the response times should be low enough such as that users are satisfied with their reports, the data from the data warehouses should always be synchronized and recalculated with the operational data, the logical server itself which holds the application should not stop working, or even the physical server itself should not have a failure (electricity, security breach etc.). Computing the performance indicator must take into account all these elements, if we are to translate the SLA.

World of technicians – or the 'underworld', apart from the world of customers, that specializes in improving the raw pure technical performances. Performance is easier to measure, as it is based on SLOs and objective indicators. For example, the lower the query response time for a data warehouse, the better the performances are, with regards to no other elements. The relation between the technical and the client world is not trivial. As we have seen, providing good technical performances doesn't necessarily provide good service

levels. It serves to nothing to have the best technical resource configuration for an application if the application is never used. Following this, there should be a constant communication between the two worlds, an element that is still missing in implementation with DSSs, and which is vital to the development of business.

6.3.1.2 Academic

This section presents the main research fields that are covered in this thesis, and describe the problematic that we treat in rapport with them. These are (directly related to industrial problematic):

- *Knowledge acquisition and formalization* – how is the non/semi-structured knowledge gathered, unified and formalized into knowledge bases, such as ontologies.
- *Autonomic adoption* – how can systems adopt autonomic behaviors and tasks can be rendered autonomic, using technologies such as Autonomic Computing
- *Cross domain* – how to combine the two research fields of Web Semantics (Ontologies) and Autonomic Computing together for DSS and DW management.

6.3.1.2.1 Knowledge acquisition and formalization

The knowledge formalization is a very live subject throughout the academic world, especially with the development of the collaborative web. We have seen in the state of the art that there is a real problem with knowledge acquisition, as we find it everywhere and in every possible form. In order to make it usable, an entire chain of processes must be elaborated, from finding the information and filtering it to what is of interest to transforming and representing it in the desired structured form. The following phases are identified:

Knowledge identification – expressing the act of locating the information. Information comes from a wide variety of sources: technical editor documents, specialized forums and discussion boards, editor guidelines and last but not least from the user experience (the harder to get and formalize). Many recent works focus on the identification of web services and their availability to perform sets of tasks.

Knowledge filtering – once the knowledge is located, there is an entire research area over how to identify what is useful and if it is trusted. Identifying what is useful depends on the objectives and on the specifications of the system, and can render autonomic some of the filtering activity. Trust on the other hand is harder to assure, and the intervention of human experts that validate the source and the information itself is often required. Trust is also the next big step for the passage to Web 4.0 (the web of confidence) so it provides an open problematic for the entire knowledge and web community.

Knowledge transformation – once the useful and trusted information has been chosen, an entire methodology is required for transforming it into a structured format (data bases, ontologies, spreadsheets etc.). Some of the functions are assured by ETL software. The data warehouse is a perfect example of knowledge transformation.

Knowledge use – the last stage, which links the formatted knowledge back with the user. In the data warehouse analogy it is basically the interface for building and accessing the rapports from the aggregated data. This is more linked to the research problematic of the human-computer interaction and to the development of graphical user interfaces.

6.3.1.2.2 Autonomic adoption

The problematic of autonomic adoption is formalized as the manual/autonomic task problematic. As research fields, this is linked with Artificial Intelligence and Multi Agent Systems, as shown in the state of the art. Yet, as pointed, the role of autonomic computing is not to replace the human but to help him with his tasks. From this perspective, the research fields revolve around how to find a suitable model and how to separate the high level from the low level tasks.

Autonomic Computing is one solution and there is an entire research field around it that has developed over the last years. Several models such as IBM's adoption model are studied, as to provide a possible industry standard. Expert Systems and Rule Based systems share the same objectives. In our case, brining autonomy to DSSs and DW management is very little studied (in research) or applied (in the industry). The real problem exists and we see it every day, as there is just too much work and too many tasks to be performed by the DSS experts. From this point of view, task identification and classification provide different research areas, as there is a question of risk management and cost/gain rapport.

There is a methodology to adding new tasks, task management and task formalization. Horn based rules offered a good starting point, and they are liable when their number remains within low limits. Once complex systems must be dealt with, there are a series of problematic that relate to rule elaboration and coherence. It remains an opened problematic, and propositions such as autonomic computing offer some guidelines for organizing the rules (i.e. the phases of the MAPE-K loop or the purposes of AC). This brings additional organization and facilitates management, but with high numbers of rules the problems persist. Nevertheless, such solutions have shown their effectiveness, and our works reinforce them as a good 'copilot' for human management.

6.3.1.2.3 Cross domain

When the two pylons of autonomic management and knowledge formalization are combined, they raise a series of new problematic questions.

IBMs specifications of autonomic computing indicate the presence of knowledge bases as the information sources for AC managers and for the AC adoption model. The model of the KB is left at the choice of the implementation. The state of the art presented the approach on how to use web semantic technologies and ontologies for AC adoptions, an approach that is taken further by the works in this thesis. We have seen that even if ontologies are a very viable solution, not much research has been done in this direction, and references are scarce. Nevertheless, they arise some of the questions that we try to answer or propagate forward for future research. These concerns:

- How ontology adoptions are implemented in an AC model, what is the backend of working with such models (i.e. for databases we have industry verified and adopted database servers).
- How ontology rules impact on the coherence of the knowledge base, the organization of the rules, the order of execution and their interdependence.
- How the AC model itself can be described with ontologies, under the form of a meta description: ontologies that describe how the AC model works, how ACMs communicate, based on knowledge bases which are in turn expressed by ontologies.
- How technical elements interconnect to express the two pylons together
- How human resources are used, as they require being both under the understanding of the managed system and knowledge experts and how business value can be added by the use of these technologies together rather than isolated.

We notice that the issues at hand vary from the technical level to the business objective level. With the technical level, research brings into discussion the adoption of standards: language specifications (such as OWL), communication protocols (such as WSAD) or model description languages (such as UML). With the business level, research turns to how business processes can be improved, how ontologies can help better describe business needs and objectives, and how can manual high cost tasks can be transformed into autonomic low cost tasks.

6.3.2 Objectives

We have seen above the main issues from the industrial and the academic point of view. Our objectives here are not to try to answer and find a solution for all of them. We want to prove that business management and BI, specifically DSS management, needs new directions from the information and knowledge point of view. We want to show that with these new directions, specific technologies such as ontologies and autonomic computing can be solid allies, and can greatly improve the level and quality of services offered to the clients.

A list of the main objectives, from the practical point of view, is shown below, as digressions are afterwards possible on each of these objectives:

- Find ways of transforming and integrating DSS management knowledge into a centralized structured form, to use it to help decision making. We have seen that knowledge comes from variety of sources, and when faced with real situations solutions are prone to human error. We want to help the human experts, by giving them already some of the answers they search in instant time. For example, instead of having to go through all the readme documents for hours to find a certain referenced bug, the expert would use the KB system to have the answer immediately. So, specifically, our first objective is *building an integrated knowledge base* which *models the DSS* and the information related to *DSS management.* This includes a *diagnostic module for DSS management* which would gain experts enormous time.

- The second objective is to provide *self-configuration for the pre-production deployments of DSSs*. Before setting up a data warehouse architecture, a list of load charge tests should be made to provide the best possible start configuration (e.g. hardware resources, parameter configuration, resource allocation etc.). The purpose is to provide a pre-production autonomic system that would test several possible configurations, by simulation of the user activity, as to find the 'best' suitable starting configuration. This provides the company a longer problem-free running time of their DWs.
- The elaboration of the second objective leads to detailing the third objective. Once a DSS is in place, intervention when something is wrong is always done manually by calling a human expert. This implies extra time and cost with problem resolution and human error prone solutions. We aim at providing two strategies: self-healing and a self-optimization. The self healing strategy implies monitoring the DSS and whenever an error is noticed trying to solve it based on the information from the knowledge base. This way, the human supervisor is only notified of the problem and the proposed solutions (and eventually would accept or reject them). He wouldn't have to loose time by being in that location and looking for the solutions. The self optimization strategy assumes that a system is already sane. By monitoring the DSS the strategy is to be able to modify continuously the resource allocations (where possible) such that they follow the various workloads and needs specified by the data warehouse usage.
- The last objective is linked to the self optimization strategy, as we want to prove that following optimization over raw technical performance indicators is not always a good thing for a DSS. We want to underline the fact that the quality of service is much more important for the business world, and that any approach to DSS management should always take into consideration this aspect first. Part of the objective, we study how the AC adoption must be changed in order to express a DSS, and how to define QoS performance indicators based on license agreements (SLA) and business objectives (SLOs).

The objectives argument the research choice towards the two domains: knowledge formalization and autonomic computing. We need both ways of expressing knowledge, brining it under structured unified forms, and using this knowledge in autonomic ways to avoid our experts waste valuable time with task execution with our clients DSSs.

6.4 Use case scenarios

This sub section contains three representative use case scenarios in relation with the presented problematic: a general scenario and two detailed specific sub scenarios. These are part of the daily activities and problems that a decisional expert is faced with, and represent areas where is room for improvement. The formalization of the scenarios is done using UML use case diagrams via the Astah UML modeling tool [Astah (2010)].

6.4.1 General use case scenario

The general use case scenario illustrates both the current working conditions for a DSS expert and how our solutions integrate with them. There are two parts, the current situation today and how this situation will improve 'tomorrow' with our solutions. The diagram is presented in Figure 46.

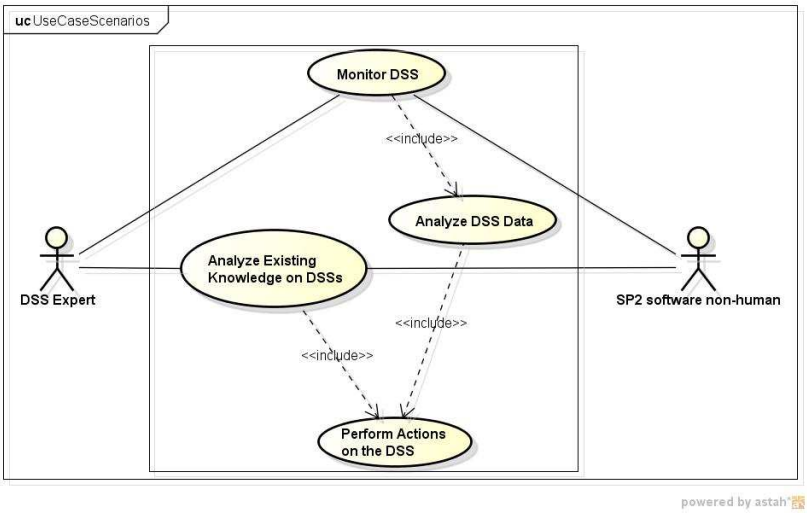


Figure 46 : General Use Case Diagram

First, the current situation today is the one including only the DSS expert actor (the left side of the diagram). DSS data observation and analysis is done by one or several DSS experts. Decisions are taken based on their analysis and the actions are performed by the same human factor. There are four general actions that can be performed, which prior to any modeling can be fitted with the phases of the autonomic computing manager:

- *Monitor DSS* – the action of observation of the DSS. It generates the totality of specific data available for the DSS.
- *Analyze DSS Data* – the analysis of the specific data collected from the monitoring.
- *Analyze Existing Knowledge on DSS* – the analysis of any existing information on DSS management, which combined with the analysis of the specific DSS data, gives the full picture over the state of the system.
- *Perform actions on the DSS* – the final action of implementing and carrying any modifications required by the two analyses.

Second, the situation of tomorrow which integrates the usage of our approach, to help DSS expert perform the presented actions. As we can see the actions remain the same. But, in addition, they integrate the autonomic software solutions which take an important part of the DSS expert's tasks.

6.4.2 Specific use case scenarios

Before describing the specific use case scenarios, we present a typical environment in which these scenarios take place. We consider a target client company: BESTCO, which is a

franchise that activates in the area of producing and selling sweets and chocolate. All the data concerning the activity of BESTCO is stored and analyzed via its information system, and implicitly its DSS.

6.4.2.1 DSS Architecture

The DSS architecture is implemented over several physical machines, in conformity with the table below:

Table 6 : BESTCO hardware DSS

Server Name	Technical Description	Software
SRV_SGBDR	8 CPU 8 Go RAM 1 TB Hard Disk	Informatica PowerCenter Oracle 9i Oracle 10g Microsoft SQL Server 2005
SRV_OLAP	4 CPU 8 Go RAM 4 TB Hard Disk	Oracle/Hyperion Essbase 9.3.1 Microsoft SQL Server 2005 Analysis Services
SRV_BO	2 CPU 2 Go RAM 1 TB Hard Disk	Business Objects XI
SRV_S9	4 CPU 8 Go RAM 500 TB Hard Disk	Oracle/Hyperion System 9 Interactive Reporting Oracle/Hyperion System 9 Financial Reporting Oracle/Hyperion System 9 Planning

There are several departments that represent the internal clients of the DSS. These departments are: Logistics, Accounting, Financial, Human Resources, Marketing and Commercial. These internal clients make use of the logical applications (groups of data warehouses) of the decision system. We consider a logical application as an application seen by the user by its utility, and not its technical modules or associated software.

We restrain the perimeter to the SRV_OLAP physical server, as it is there where the data warehouses are physically stored; the table below presents its organization:

Table 7 : SRV_OLAP organization

Logical Application	Internal Clients (departments)	Utilization Period	Logical Server	Analytical Physical Application / Base
Budget elaboration and previsions	Logistics Accounting Financial HR Marketing Commercial	Each day from 15/08 to 15/12	Essbase	BUD/LOG BUD/ACC BUD/FI BUD/HR BUD/MKG BUD/COMM
Financial Reporting	Financial	Each month between the 1st and the 5th during day	Essbase	FIN/FI
Cost analysis	Financial Accounting	Each month between the 5th and the 20th during day	Essbase	CST/FI CST/ACC
Sales analysis	Commercial Financial	Each night	Essbase	SA/COMM SA/FI

The first column indicates the concerned logical applications for the SRV_OLAP. There are four logical applications: budget, financial, cost and sales. Each of these four logical applications are used by the internal departments at BESTCO for their decision making tasks (second column). In the third column we specify the periods in which each of these logical application is used. The fourth column contains the associated logical server, in this case the Essbase OLAP server. Finally, in the last column we have the Essbase physical applications and their corresponding Essbase bases. We considered that each logical application has one corresponding physical application (i.e. Budget elaboration and previsions – BUD; Financial Reporting – FIN etc.).

6.4.2.2 DSS Management

BESTCO has several persons in charge of taking decisions (decision responsible), who are the users of the DSS. Their organization is as follows:

- 1 DSS responsible with full rights to the entire DSS. He can modify the configuration parameters and the organization of the data warehouses, and is responsible for their well functioning
- 1 PDG who is the general manager of BESTCO and who accesses reports from all departments
- 2 budget decisional users in charge of the audit for the budget application. They require reports from the budget logical application
- 1 financial reporting decisional user who analyses the financial data from the financial analysis logical application
- 1 cost decisional user responsible for the analytical data related to the cost application
- 2 sales analysis users in charge of the sales analysis

6.4.3 Punctual resource configuration

The punctual resource configurations is a case scenario where given a certain configuration and a specified date, we change the parameter configuration so that better performances are achieved. It is a configuration on demand scenario.

The situation is the following: The budget users notice that their reports (data retrieval operations) on the budget application take more time than usual to generate and they are not content with these times. This happens on the date of 23.09 at 11am (therefore during day time). They call the DSS responsible and demand what is wrong. The DSS responsible will in turn analyze the environment and see that due to the increase in the size and activity of the budget application bases, a reconfiguration of parameters is required, specifically the reconfiguration of cache memory allocations. So, based on the current situation and environment, given rules and personal experience, the DSS responsible will connect to the SRV_OLAP and change the cache parameters. The use case scenario is described step by step, from the point of view of the DSS expert in Figure 47.

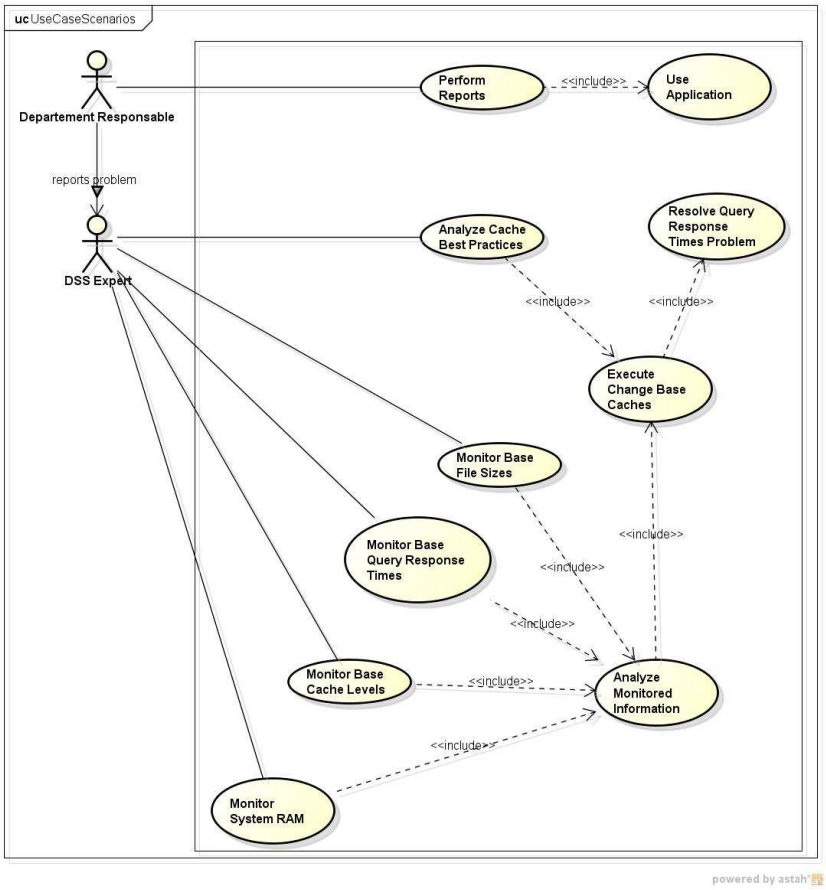


Figure 47 : Punctual Resource Configuration Use Case

6.4.3.1 Step 1

The environment and configuration for the date of 23.09 at 11am is observed by the DSS expert who constructs the following table:

Table 8 : SRV_OLAP punctual resource configuration scenario

Logical Application	Utilization Period	Physical Application / Base	Index File Size (Go)	Data File Size (Go)	Index Cache (Mo)	Data File Cache (Mo)	Data Cache (Mo)	Access mode	Storage Mode
Budget elaboration and previsions	Each day from 15/08 to 15/12	BUD/LOG	10	100	100	300	10	Direct	BSO
		BUD/ACC	15	200	200	500	50	Direct	BSO
		BUD/FI	8	150	200	300	30	Direct	BSO
		BUD/HR	0,1	0,3	10	20	3	Direct	BSO
		BUD/MKG	1	12	100	200	10	Direct	BSO
		BUD/COMM	10	80	80	100	50	Direct	BSO
Financial Reporting	Each month between the 1st and the 5th during day	FIN/FI	15	300	500	N/A	1000	Buffered	BSO

Cost analysis	Each month between the 5th and the 20th during day	CST/FI	5	100	300	N/A	500	Buffered	BSO
		CST/ACC	3	80	200	N/A	400	Buffered	BSO
Sales analysis	Each night	SA/COMM	1	100	200	N/A	200	Buffered	BSO
		SA/FI	2	130	200	N/A	250	Buffered	BSO

6.4.3.2 Step 1

The following rules are taken into account by the DSS expert when changing the cache allocations:

- *Rule 1*: The minimum values of the caches must be respected (from the Essbase 9.3.1 technical readme file)

Table 9 : Minimum cache values

Type de cache	Access Mode	Minimum (KB)
Index Cache (IC)	Buffered	1024
	Direct	1024
Data File Cache (DFC)	Buffered	N/A
	Direct	10240
Data Cache (DC)	Buffered	3072
	Direct	3072

- *Rule 2:* The sum of the allocated caches for all the bases on SRV_OLAP must not be grater then the amount of RAM memory installed (from the Essbase 9.3.1 technical readme file)
- *Rule 3:* In direct access mode the data cache should have a value of 12,5% of the data file cache (from the Essbase 9.3.1 cache configuration advice technical web documentation)
- *Rule 4:* During day, the cache allocation should be: 70% index cache, 30% data cache and data file cache (from the DSS expert knowledge)
- *Rule 5:* During night, the cache allocation should be: 20% index cache, 80% data cache and data file cache (from the DSS expert knowledge)
- *Rule 6:* A logical application in its utilization period should have 5 times more RAM memory allocated into caches than a logical application which is not critical (from the DSS expert knowledge)
- *Rule 7:* Two logical applications on the same utilization periods with the same priority are allocated the same amount of RAM memory in proportion with their sizes (from the DSS expert knowledge)
- *Rule 8:* The general cache allocation rule is: allocate the maximum possible RAM memory into caches (from the Essbase 9.3.1 technical readme file):
    - Index File Cache = Index File Size (ideally)
    - Data File Cache + Data Cache = Data File Size (ideally)

- *Rule 9:* The allocated memory for a logical application is divided between its bases proportional with the file sizes of each base and with regard to Rule 1 (from the DSS expert knowledge)

6.4.3.3 Step 3

The DSS expert builds a table containing the RAM memory needs for each base and logical application, for the date of 23.09. The bases inherit the utilization periods from their corresponding logical application.

Table 10 : RAM Memory occupation and needs on SRV_OLAP

Logical Application	**Physical Application / Base**	**Is in Utilization Period?**	**Needed RAM (Go)**	**Occupied RAM (Mo)**	**Total Occupied RAM (Mo)**
Budget elaboration and previsions	BUD/LOG	Yes	110	410	2263
	BUD/ACC	Yes	215	750	
	BUD/FI	Yes	158	530	
	BUD/HR	Yes	0,4	33	
	BUD/MKG	Yes	13	310	
	BUD/COMM	Yes	90	230	
Financial Reporting	FIN/FI	No	315	1500	1500
Cost analysis	CST/FI	No	105	800	1400
	CST/ACC	No	83	600	
Sales analysis	SA/COMM	No	101	400	850
	SA/FI	No	132	450	

So there is a total of 6013 Mo of RAM memory that is used from the 8Go available, which indicates that there are still 1487 Mo free RAM to allocate (500Mo are reserved by the operating system).

6.4.3.4 Step 4

The DSS expert applies the set of rules with the RAM memory occupation and needs. He first computes the amount of memory each logical application should occupy, based on rule 1, 2, 6, 7, 8 and 9. The table he builds is:

Table 11 : New Allocated RAM memory

Logical Application	**Physical Application / Base**	**Allocated RAM (Mo)**	**Allocated RAM (Mo)**
Budget Elaboration and previsions	BUD/LOG	1172	6250
	BUD/ACC	2291	
	BUD/FI	1684	
	BUD/HR	14	
	BUD/MKG	139	
	BUD/COMM	950	

Financial Reporting	FIN/FI	416	416
Cost analysis	CST/FI	232	416
	CST/ACC	184	
Sales analysis	SA/COMM	180	416
	SA/FI	236	

6.4.3.5 Step 5

Once the new RAM allocation values are determined, the DSS expert uses rules 1, 3 and 4 to determine the new index, data and data file cache values for each the bases and application. The table he builds is:

Table 12 : New cache allocation for the Essbase bases on SRV_OLAP

Logical Application	**Physical Application / Base**	**Allocated RAM (Mo)**	**Index Cache (Mo)**	**Data File Cache (Mo)**	**Data Cache (Mo)**
Budget elaboration and previsions	BUD/LOG	1172	821	317	35
	BUD/ACC	2291	1604	619	69
	BUD/FI	1684	1179	455	51
	BUD/HR	14	1	8	3
	BUD/MKG	139	97	38	4
	BUD/COMM	950	665	257	29
Financial Reporting	FIN/FI	416	291	112	12
Cost analysis	CST/FI	232	163	63	7
	CST/ACC	184	129	50	6
Sales analysis	SA/COMM	180	126	49	5
	SA/FI	236	165	64	7

6.4.3.6 Step 6

With the new cache values computed, the DSS expert proceeds to introduce the new cache values into the base configurations. To do so, he uses the Essbase Management Studio Interactive GUI and set of Essbase scripts he has developed to this end.

With the new configuration in place, we can see that the problematic logical application has now reserved most of the RAM memory as it is the only application in utilization period. The DSS expert reports to the two responsible of the budget application that the new configuration is in place such that they can continue using the application, without the problem of long reporting times.

6.4.4 Continuous resource reallocation

The continuous resource reallocation scenario addresses the issue of system self-management over the self-optimization principle of Autonomic Computing. Unlike the punctual resource configuration, this case scenario treats the aspects of continuous

configuration. Future configurations are built from past feedback rather than from predetermined configuration rules.

The situation is the following: The budget decisional users are observing that their reports on the budget application take progressively more time with each passing day of the week, throughout the month of September. On the day of Friday, 23.09, the budget decisional users are forced to call the DSS expert to do a reconfiguration of the resource allocation (the punctual resource allocation scenario). The DSS expert performs a reconfiguration of the resources using the rules from the punctual configuration scenario. Even if the query response times improve, the budget decisional users are still not happy with the report times, as they feel this could further improve. They base their argument on the fact that in the previous years during the same month reports were generated faster. The DSS expert knows that the rules he applied for the punctual configuration suffice no longer, so it adopts a different approach for configuring the caches. He proposes the usage of heuristics with an autonomic system that monitors the performances each day, and changes the values of the caches in concordance with the previous days through intelligent feedback loops (the BIOptim approach).

First, the DSS expert performs a punctual resource configuration and then explains to the budged experts that even if it will take some time before the budget application will deliver improved reporting times, the feedback mechanism will allow, once learned, to always have a high level of performance. The decisional experts agree to allow this small performance sacrifice with regards to future expectations. The scenario is illustrated in Figure 48.

6.4.4.1 Step 1

The DSS expert performs a punctual resource configuration for the date of 23.09 (which is considered Day 0 for the continuous resource configuration). Therefore it starts with the configuration from Table 13.

Table 13 : The Budget Application after the punctual resource configuration at Day0

Logical Application	**Physical Application / Base**	**Allocated RAM (Mo)**	**Index Cache (Mo)**	**Data File Cache (Mo)**	**Data Cache (Mo)**	**Average Query Response Time (s)**
Budget elaboration and previsions	BUD/LOG	1172	821	317	35	7
	BUD/ACC	2291	1604	619	69	4
	BUD/FI	1684	1179	455	51	4.8
	BUD/HR	14	1	8	3	8
	BUD/MKG	139	97	38	4	5.5
	BUD/COMM	950	665	257	29	4.9

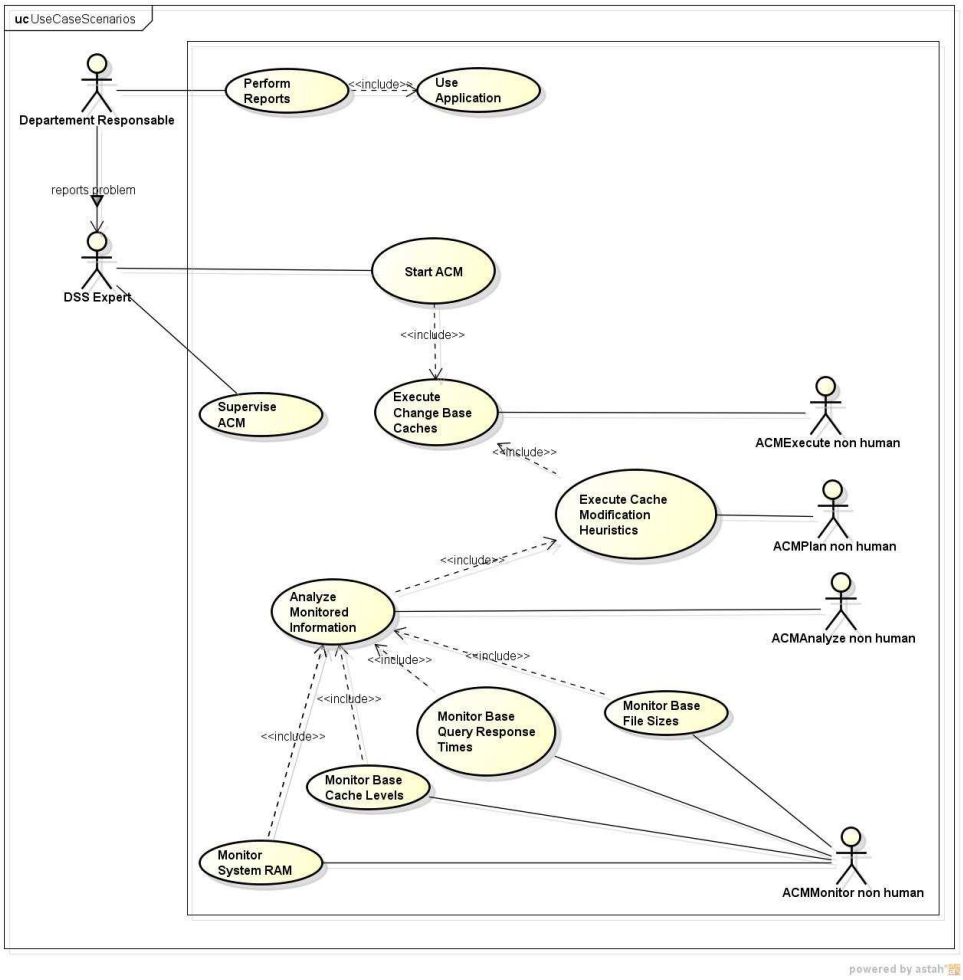


Figure 48 : Continuous Resource Allocations Optimization with Autonomic Computing Use Case

6.4.4.2 Step 2

The DSS expert sets the autonomic computing system and strategy for system self-improvement. The following steps are performed by the autonomic system BIOptim and no longer by the DSS expert. There are two different autonomic computing managers, one for the BUD physical application and the other for each of the bases of the BUD application. Each autonomic manager is detailed with its MAPE-K loop as follows.

Monitor (Bases): Table 14 shows the monitored elements for the budget application on SRV_OLAP. The first column describes the type of the element (an entity of the DSS, a configuration, resource or performance indicator). The second indicates the code/name of the element. The measure column contains, where the case, the measure unit of the element. The last column contains the source from which the element and its measure are extracted. In this case we are faced with three different sources:

- SQL DBs for the architectural organization (obtained with the help of SP2s BICopilot software).

- .log files that contain the system resource information for the RAM memory allocation
- .log files from the Essbase server containing the file sizes, cache values and retrieval time values. The monitored elements are considered per base, even if they are extracted from a single common log file for the BUD physical application.

Table 14 : Monitored elements over the SRV_OLAP server for the budged application

Type	Name	Measure	Source
Physical Application	BUD	N/A	dbo.ACMDB_OPTIM_ MBG_AggPerfStep
Base	LOG	N/A	
	ACC	N/A	
	FI	N/A	
	HR	N/A	
	MKG	N/A	
	COMM	N/A	
Resource/ server	Available RAM memory	Mo	PerfSys_Monitor.log
Measure/ base	Index File Size	Go	Essbase/logs/app/ BUD.log
	Data File Size	Go	
Configuration/ base	Index Cache	Mo	
	Data File Cache	Mo	
	Data Cache	Mo	
	Access Mode	String	
	Storage Mode	String	
Performance/ base	Query Retrieval Time	s	
Resource,Measure, Configuration, Performance/Physical Application,Server	Aggregated Element	N/A	dbo.ACMDB_OPTIM_ MBG_AggPerfStep

6.4.4.3 Step 3

Monitor (BUD): BIOptim performs aggregation operations on the monitored base indicators, such that they are expressed also for the BUD application. The aggregation is done differentially in function of the concerned indicator, as follows:

- Aggregation can only be done on quantity measures (e.g. you can aggregate the Index Cache but not the Access Mode)
- Resource, Measure and Configuration indicators aggregate with '+'
- Performance indicators aggregate with 'AVGERAGE'

After the aggregation operation, the following line is added to the Monitor table:

Table 15 : Aggregation for the BUD application on SRV_OLAP

Logical Application	Physical Application / Base	Allocated RAM (Mo)	Index Cache (Mo)	Data File Cache (Mo)	Data Cache (Mo)	Average Query Response Time (s)
Budge elaboration and previsions	BUD AGGREGATE	6250	4367	1694	191	6.33

6.4.4.4 Step 4

The system performs two improvement heuristics, one for each base and one for the BUD application. For the base heuristics, the system tries to decrease the sizes of the Index Cache each day with a given Δ, checking that performance differences between the current day and the previous ones are not grater than a threshold β. This corresponds to the analysis phase of the ACM.

Analyze (Base), Table 16 presents the new cache and response time values for the day of 24.09 (Day 1), with Δ = 0.05 and β = 0.05.

Table 16 : Budget application configuration analysis for the date of 24.09 (Day 1) in rapport with the previous days

Physical Application / Base	Index Cache prev. (Mo)	Index Cache (Mo)	Freed Mem. (Mo)	Avg. Query Time prev. (s)	Avg. Query Time (s)	Perf. Difference	Is Change Accepted (<=β)
BUD/LOG	821	738.9	82.1	7	7.5	0.07	No
BUD/ACC	1604	1443.6	160.4	4	4.2	0.05	Yes
BUD/FI	1179	1061.1	117.9	4.8	4.9	0.02	Yes
BUD/HR	1	1	0	8	8	0.00	Yes
BUD/MKG	97	87.3	9.7	5.5	5.7	0.04	Yes
BUD/COMM	665	598.5	66.5	4.9	5.5	0.11	No

Note. For the HR base the index cache value doesn't change according to Rule 1 (regarding the minimum cache values).

Taking into consideration the β acceptance levels, the index cache configuration table for the analysis of BUD for 24.09 is show in Table 17. A total of 288 of Mo have been saved, whereas the aggregated averages query response time increased by 0.014.

Table 17 : Analysis output for the BUD bases for the index cache on Day 1

Physical Application / Base	Index Cache (Mo)	Freed Memory (Mo)	Average Query Response Time (s)
BUD/LOG	821	0	7
BUD/ACC	1443.6	160.4	4.2

BUD/FI	1061.1	117.9	4.9
BUD/HR	1	0	8
BUD/MKG	87.3	9.7	5.7
BUD/COMM	665	0	4,9
AGGREGATION	4079	288	5.78

Plan and Execute: the new index cache values are made permanent for the bases of the BUD application, for the date of 24.09 (Day 1).

6.4.4.5 Step 5

Analysis (BUD) Passing on to the next day of 25.09 (Day 2), BI Self-X runs the application heuristic over the BUD application. This will reallocate the freed memory from the base heuristics to the non performing bases. The following two aspects are considered:

- A base is performing if its average query response time is lower than its application average (BUD -5.78s); otherwise it is considered non-performing.
- The reallocation of the freed memory (288 Mo) is done equally between the number of non performing bases (LOG and RH)

The situation before the reallocation is shown in Table 18. There are two non-performing bases: LOG and HR.

Table 18 : Analyze BUD for memory reallocation

Physical Application / Base	Index Cache (Mo)	Average Query Response Time (s)	Performing
BUD/LOG	821	7	No
BUD/ACC	1443.6	4.2	Yes
BUD/FI	1061.1	4.9	Yes
BUD/HR	1	8	No
BUD/MKG	87.3	5.7	Yes
BUD/COMM	665	4.9	Yes
AGGREGATION	4079	5.78	N/A

After the reallocation each of the two non performing bases acquire 144 Mo. The new index cache value are shown below for the two bases, LOG and HR:

Table 19 : Non performing BUD bases free memory reallocation into index cache for Day 2

Physical Application / Base	Index Cache (Mo)	Average Query Response Time (s)
BUD/LOG	965	6,5
BUD/HR	145	2.2

Plan and Execute: with the final values computed, using VBS scripts and the Essbase console, the new index cache values are made permanent for the BUD application, for the date of 25.09 (Day 2).

6.4.4.6 Step 6

The DSS expert decides the autonomic strategy by mixing the two heuristics, using them over a period of 2 months, from 23.09 to 23.11. The base heuristics is run each day, where as the application heuristics once each 5 days. Moreover, the two heuristics exclude each other. They can't run both in the same day, and the application heuristic has the priority. The running chart is shown in Table 20, for a period of 15 days:

Table 20 - BUD Application heuristics between 23.09 and 07.10

Day	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7
Base Heuristics		X		X	X	X	X	X		X	X	X	X	X	
BUD Heuristics			X						X						X

At the end of the optimization period, a good ratio between cache allocations and query response times is achieved, satisfying the demands of the budget department responsible.

6.5 Conclusion

This section presented: the context of the thesis. First, we have presented the industrial environment, with the SP2 Solutions company. Then, the main issues from the point of view of the industrial and the academics worlds were described. They focused around two central themes: knowledge management and task management. Consequently, in concordance with these issues a series of objectives has been elaborated for the thesis, with the central solution: BI Self-X.

In order to better exemplify some the presented issues, three case scenarios were described: a general one and two specific. They illustrated how knowledge management and task management impacts the functioning of a DSS, and they were based on our experience with our clients. Moreover, the usage of the BI Self-X improvement heuristics has shown a part of how our solutions can help improve the current situation with DW management.

The presented scenarios are part of a much larger spectrum of possible problems. Our aim was to underline the fact that DSSs are faced with them in the real world, and that some of our propositions can help the DSS experts manage their system.

7 The ‘BI Self-X’ Approach

“If I'd had some set idea of a finish line, don't you think I would have crossed it years ago?”

B. Gates

7.1 Introduction

Starting from the technologies presented in the state of the art and based on the described problematic, needs, objectives and use case scenarios we elaborate further the proposed approach.

The central project is called BI Self-X and is split into three parts, corresponding to the problematic and the objectives. These are:

Decision Support System Modeling – Static – the representation of a DSS into a unified and standard form for better information access and further for autonomic adoption. This derivates from the problematic of IT systems modeling. Modeling a DSS implies describing its resources and elements, such as: physical machines, logical architectural organization, resource allocation, configuration parameters, performance indicators etc.

Autonomic Modeling – based on the modeling of the DSS , the autonomic model describes how the system should be enhanced and what modules are required for autonomic adoption. These include: the managed elements, the autonomic managers, the MAPE-K loop phases, the communication protocol between the ACMs etc.

Knowledge Base Modeling - Dynamic – this last part focuses on a model for storing the entire information needed for managing the DSS with the support of the Autonomic Computing model. It is actually the bridge between the DSS and the autonomic modeling. It includes the knowledge that is in the center of the autonomic computing managers, the analysis rules for assuring the four principles of the AC, the best practices and pieces of advice that decisional experts use to manage the data warehouses, and the self-management heuristics.

The presentation of the proposed approach is split into two main sections: UML modeling and adopted technologies (ontologies and autonomic computing) modeling. The UML model shows our vision over how the DSS and the information for its management can integrate. All of the three aspects (Static, Autonomic and Dynamic) are shown to fully understand our model. Afterwards, the adopted technologies modeling transforms the UML into an ontology and autonomic computing model, following several transformation guidelines, and equally introducing new elements (such as the non structured information formalization).

7.2 UML modeling

This section presents the totality of the UML models in our system. This offers a clear illustration of the proposed approach.

7.2.1 DSS modeling - static

Modeling a DSS derives from modeling an IS. It should contain all elements that are part of the system and concern its functioning, such as: the system's physical and logical elements, the resources it needs to function under various conditions, the configuration indicators, the performance measures, the users, the system activity etc. This is the static modeling because the elements and information that is modeled in this section does not change. Any particularities of a specific DSS, such the decision software solution adopted or the logical / physical architecture can be fitted into one of the elements described with our model. An important thing to keep in mind, while going through the next modeling sections, is that the choice of software used for the exampled is purely based on our prototype and experiments (Oracle Essbase). Nevertheless, the model is thought to be generic, therefore any DSS solution should fit into it.

In the light of what a DSS should represent we describe our modeling proposition in Figure 49.

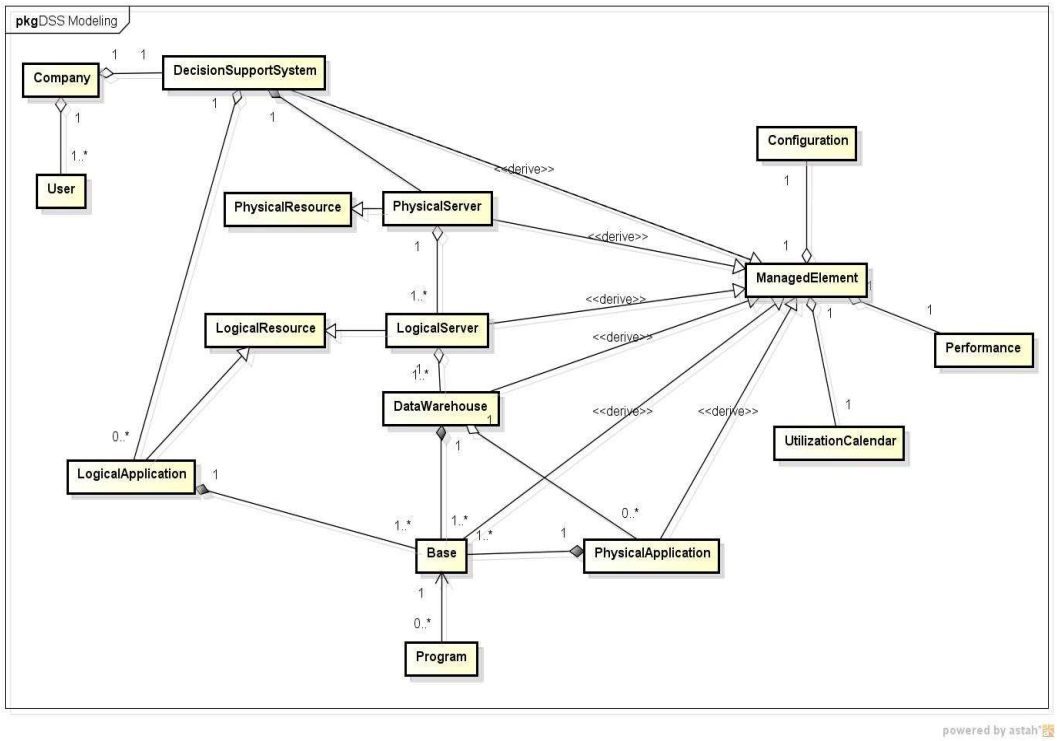


Figure 49 : DSS Model Diagram

In the overall schema we can distinguish three sub-parts:

- *External* – the external factors that are related to a DSS. This includes the users of the DSS and their roles according to usage policies, the company and the software editors.
- *Managed Elements* – the elements of the DSS architecture and how they are interconnected

- *Element Properties* –the properties of the managed elements, the performance indicators and the logical and physical resources used for each of the managed element

7.2.1.1 External

The External management component contains the external factors that are related or directly implicated in using the DSS. The detailed schema is shown in Figure 50.

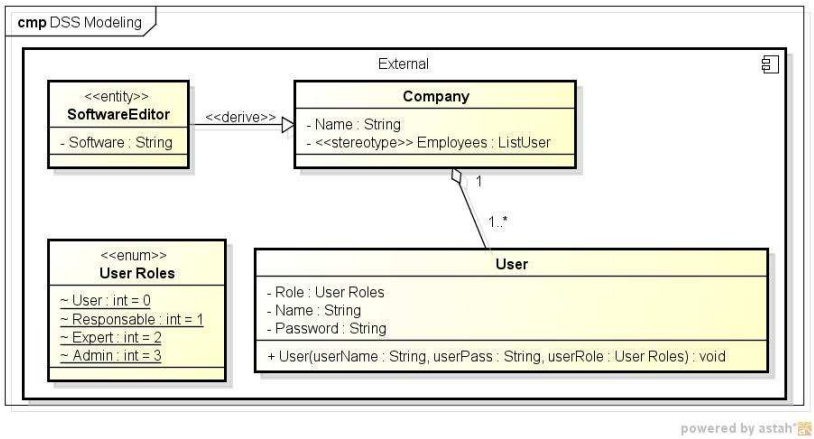


Figure 50 : External management diagram

Software Editor –contains the software editors (such as Oracle, Microsoft, IBM etc.). There is a link of relation between this class and the Company class, as a Company can be a Software Editor itself or it can have agreements with other software editors for using their products. The attributes of the Software Editor class are:

- *ID (string)* – the unique id of the editor: e.g. Microsoft Corporation
- *Name (string)* – the used name for the company, which in most of the cases can be the ID itself: e.g. Microsoft

Company –contains the company that is concerned with the decision support system. A company is related to other software editors, has a decision support system (from the Decision Support System class) and has a number of employees, from where the 1..* relation with the User class. The attributes of a Company are:

- *ID (string)* – the unique id of the company: e.g. FTC
- *Name (string)* – the used name for the company: e.g. France Telecom
- *Employee (string)* – the employed personnel, as an instance of the User class.

User –describes the human users of the DSS. They are usually employees of the company or third party persons (i.e. external experts) hired by the company ass DSS experts. A user is equally connected by a use relation to the Decision Support System class. An instance of the User class contains the following attributes :

- *ID (string)* – the unique id of the user: e.g. JMAY

- *Name (string) / Password (string)*– the credentials of the person: e.g. John Mayor / jmpass
- *UserRole (User Roles)* –the role of the user, from several possible roles described in the User Role class. A user can have only one role at a given time.

User Roles – the class contains the list of roles that a user can have. This is an enumeration of the defined roles. The role of a user defines his usage scenarios, his 'responsibility' areas and security policies. The usage scenarios vary from simple interpretation of the generated reports to administration and system maintenance operations. There are several specified roles:

- *User (constant int = 0)* **–** the basic user role that is the capability of using the decision system for generating reports that are interesting. In this role no modification to the data or information is allowed, as it is a read only access.
- *Responsible (constant int = 1)* – usually the responsible of a department such as commercial, budget etc. The responsible has the same rights as the user, with the addition that he can generate and save a certain number of analytical reports that may of interest to the common user. This way, when a user requests the specific rapport, the answer comes from the pre-generated rapport in practically an instant and doesn't charge the decisional system.
- *Expert (constant int = 2)* – this represents the decisional expert, with all the right of the responsible and also with the capabilities of modifying system parameters (such as resource allocations), executing actions such as data recalculation and restructuration, modifying the existent data warehouse architecture etc.
- *Admin (constant int = 3)* **–** it is the traditional 'root' privileged user that has control over all the system. Usually this is reserved to the ones in charge of the respective company and its decision support system.

7.2.1.2 Managed elements

The managed elements represent the entities that form the DSS architecture. They are organized hierarchically following the Inmon data warehouse model as shown in Figure 51. There is a totality of six elements, from the highest levels of the hierarchy (the Decision Support System) to the lowest (Bases).

Decision Support System – represents the highest level of the architectural hierarchy and it sums the totality of the managed elements that are part of the decision system. Being an end point for the aggregation operations, the indicators are very general, and decisions are very hard to take directly without descending to the lower levels. In addition most of the indicators here only exist because of the lower levels, very few are DSS specific. For instance, a query response time indicator at the DSS level has no meaning by itself. Its meaning is that it is the aggregated value of the query response times from the Base levels. This is why a DSS has few specific properties:

- *ID (String)* – the ID of the element, a common property for all the managed elements. It represents an unique identifier for referring to the DSS (SP2_DSS).

- *Name (String)* – the name of the DSS associated with the respective ID. It is not unique. Several different names can be associated to the same ID. (e.g. SP2 Solutions Decision System). This too is a common property for all the managed elements.

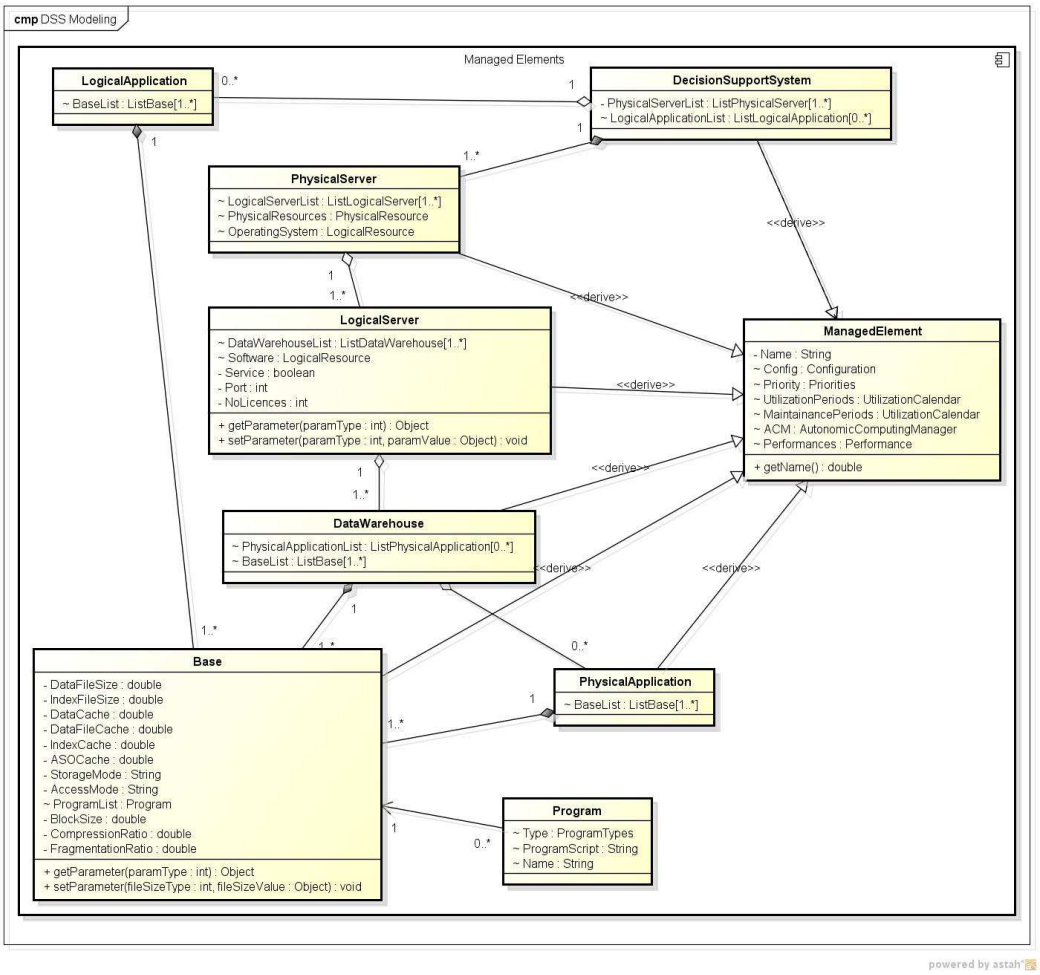


Figure 51 : Managed Elements diagram

As the ID and the Name are common properties for all the managed elements, they will not be detailed again. The same note is valid for the properties which derivate from the ManagedElement class and which are valid for all the managed elements.

Physical Server –is the formalization of the real physical machines that are used for hosting the BI software products. These machines are powerful servers as the performed operations are very resource and time consuming. A physical server can serve to many purposes, such as: handling the data base/ data warehouse architecture (very high RAM memory and disk capacity requirements); handling the front end of the GUI interfaces for report generation (powerful processor requirement); handling the archive and backup operations (a high disk storage capacity and rotation speed for fast I/O operations). The list of concerned properties is shown below:

- *The Phyiscal Resources (PhysicalResource)* **–** the concerned physical resources

    - *IP (String)* – the IP of the server for network identification
    - *RAM (int) (Mo)* – the amount of RAM memory installed on the server
    - *Level 0 Cache (int) (Mo)* – the amount of level 0 cache that the processor of the physical server contains.
    - *HardDisk (double) (Mo)* – the capacity of the hard disks connected to the server
- *OperatingSystem (LogicalResource)* – the OS installed on the server
- *LogicalServerList (list:LogicalServer)* – the list of logical servers installed on the machine (e.g. Web Server, Data base server, OLAP server, etc.)

A discussion is in place with virtualization over physical servers. Virtualization refers to the capability of sustaining several independent machines on the same physical machine. It is often used for developing and testing environments. For instance, to test how a system upgrade would impact an existing system, a virtual environment duplicating the real environment. Nevertheless, as we model decision systems architecture we eliminate this case, mainly due to performance considerations. Virtual environments demand a high resource usage and the bottlenecks are usually hard disks and RAM memory. A data warehouse physical server is very resource consuming, requiring both fast hard disk accesses and a lot of RAM memory. For this reason, we consider that a physical machine implies one physical server only.

Logical Server – a physical server can run several logical servers. A logical server is the next hierarchical entity that is in charge of offering specific functionalities (i.e. Web Servers, Data Base servers, VPN servers etc.). It is installed on the physical server's OS. The OS itself is not considered a logical server. Examples of logical servers are: Apache Tomcat Server, Open VPN Server, Oracle Hyperion Essbase etc. Similar functionalities are usually regrouped by the same logical server (i.e. an Apache Server will allow both http file publishing and php interpretation). For DSSs a logical server is the BI suite installed on the machine. The Oracle Hyperion Essbase logical server is in charge of managing the Essbase cubes and physical applications that form the data warehouse architecture from our examples. A logical server has the following properties:

- *Software (LogicalResource)* – the software that runs as the logical server (e.g. Essbase 9.2)
- *Service (Boolean)* –if it is installed as a runtime service
- *Port (int)* – the communication port number used for message exchange
- *Managed elements inherited properties* – these are valid for all the managed elements (as a derivate of the ManagedElement class). The properties below concern the usage periods and priorities of the managed element:
    - *UtilizationPeriod (Periods)* – the usage period specified as instances of the Period class (one can have several utilization periods)
    - *MaintenancePeriod (Periods)* – the specification of the maintenance period, when the logical server is running but it is not used for production (for test, load or maintenance operations).
    - *Priority (Priorities)* – the level of priority of the logical server, when in a group of several servers running on the same physical machine. This is

usually used to determine how resources are allocated between the logical servers or how performance and quality of service issues are treated.

Physical Application – the next managed element in the hierarchy, as part of the logical server. The notion of physical application relates to the grouping of several bases that share the same purpose for their data. This must not be confused with the notion of Logical Application. A logical application indicates only the formalization and regrouping of elements that share the same objectives, and not their physical storage. The physical application doesn't condition the existence of bases. It merely physically regroups the bases under specific organizations. For example, on a physical server we have several OLAP cubes that contain data regarding the budget of an enterprise. Some other cubes contain data for the marketing department. As these cubes are stored on the same physical machine, a difference between the two groups is done by organizing them under two different physical applications (one for the budget related cubes and one for the marketing related cubes). This is why the relation with the bases is 1..*. A base can only physically belong to a single physical application. The only specific property of a physical application is the list of bases:

- *BaseList (list:Base)* – the list of bases that are contained by the physical application

Base – represents the core of the information system, as it is the entity that contains the actual data. The base is the lowest class of the managed elements hierarchy and it formalizes any way of storing the analytical data. This includes data bases, multidimensional bases, OLAP cubes or data marts. The term Base is generic (not to be mistaken with data base). With DSSs the representations used are usually multidimensional bases and OLAP cubes. As the lowest hierarchical entity, the measurement and performance indicators are raw and not aggregated. As seen in the diagram, there are a higher number of specific properties compared to the other managed elements. Some of the properties described below, such as the index file cache, data file caches, ASO, BSO are particularly related to Essbase (as described in their technical documentation). When using other technologies, these either have correspondences (i.e. different names for different products from different editors for the same functionalities) or don't exist.

- *StorageMode (String)* – the way in which the data in the bases is organized and stored for persistence mechanisms. With an Essbase base, there are two ways of storing data: block (BSO) and aggregate (ASO). The BSO represents multiple block storage of the bases that have no hierarchy, and is usually suitable when dealing with both data access and data modification. The ASO is an alternative to the BSO, by enabling aggregation storage with hierarchies. ASO is much faster for aggregation, dimension scalability and data access, and is usually employed when the accesses to the bases are read-only. With ASO only one base per physical application is allowed.
- *AccessMode (String)* – defines the way the data from the bases is accessed. There are two ways of accessing data: Direct I/O and Buffered. Direct I/O access means that data is accessed directly by looking into the physical data file on the hard disk. Direct I/O requires fast hard disks, and is suitable for both read and write data operations. Buffered access means that the data is accessed from specific cache buffers, which store (in duplicate) some of the data from the bases (e.g. the recently

demanded data from the base). The buffered access is much faster, and it is used for read only operations. There is a smaller charge on disk operations at the cost of an increased usage of RAM memory.

- *DataFileSize (double) (Mo)* – the space occupied on the hard disk by the data file that physically stores the base data. It is also known as the page file size. Each time new data is added to the base, the data file size increases
- *IndexFileSize (double) (Mo)* – the space occupied on the hard disk by the index file, used for indexing the base data with the purpose of faster access. For a multidimensional cube, each time a new dimension is added, the index file increases accordingly.
- *IndexCache (double) (Mo)* - The index cache is a memory buffer that holds index pages. Heir number depends upon the amount of memory allocated to the cache. With Essbase this is used only with the BSO storage mode.
- *DataFileCache (double) (Mo)* -The data file cache is a memory buffer that holds compressed data files (.pag files). Essbase allocates memory to the data file cache during data load, calculation, and retrieval operations, as needed. It is used only with the BSO storage mode and when direct I/O is in effect.
- *DataCache (double) (Mo)* - The data cache is a memory buffer that holds uncompressed data blocks. Similar to the data cache it is used only with BSO storage mode.
- *ASOCache (double) (Mo)* – The equivalent of the three caches in BSO is the ASO Cache in ASO. The allocation depends on the RAM memory and on the Level 0 cache values.
- *BlockSize (double) (Ko)* – The dimension of a single block from the multidimensional cube. A block represents the 'atom' of the data cube, and by varying its dimension actions such as data retrieval or cube calculation are influenced.
- *CompressionRatio (double)* – relates to the fraction of the base which is uncompressed in rapport with the full size of the base.
- *FragmentationRatio (double)* – refers to the level of fragmentation of the data files. As these files are stored physically on hard drivers, similar to any files they defragment over time, slowing down accesses, thus impacting data retrieval operationsl.

Program – a program is a succession of instructions that enables specific operations on a managed element, actions such as: data recalculation and aggregation, data retrieval, reporting, or data export. In our model, the program is attached only to bases, as we employ only base specific programs. A base can make use of several programs. The sole specific property of a program is its type.

- *ProgramType (Program Types)* – the type of the program, from the types defined in the Program Types class in the Element Properties.

Logical Application –represents a regrouping of several managed elements that share the same objective. It is an application as seen by the user of a decisional system without taking into account the different technical aspects. This is why its links are between the

highest level architecture (the DSS) and the lowest level (the Base). Consider a user interface used for accessing reports. The GUI application is in this case a logical application as it connects to all the data sources and returns the required report. It can thus access bases from several different physical servers, different applications etc. A logical application doesn't have a clear status as placing it in the hierarchy and there are no specific characteristics of a logical application.

Data Warehouse – the data warehouse regroups in our model the organization of physical applications and bases (the elements that organize the stored data). It is not a managed element itself as a data warehouse is, by its definition, an architecture of the stored data. We can have several data warehouses depending on the needs and specifications that can include any number of bases and/or physical applications. For example we can have an archive data warehouse that contains the archived data (i.e. data that is older then 3 months). Or, we can have a data warehouse that has only one base used for user information. The specific properties of a data warehouse are the list of bases and physical applications.

7.2.1.3 Element properties

The last sub diagram contains the resources, properties and types that describe the managed elements. The performance and configuration indicators aggregate over the DSS architecture hierarchy. The Element Properties class also contains also the enumerations of constant values, such as Priorities or Program Types.

To exemplify how aggregation indicators work over the managed elements, consider a Base that occupies a certain amount of disk space on the physical drive. The total amount of disk space is given by the sum its file sizes (Data and Index). This sum identifies the indicator of base disk occupation. Even if this indicator is base specific, by aggregating it (using addition) we can refer to physical application disk occupation or logical server disk occupation. This way a specific managed element indicator becomes an overall description indicator.

Figure 52 presents a detailed view over the Element Properties class, making a distinction between the resources and performance/configuration indicators.

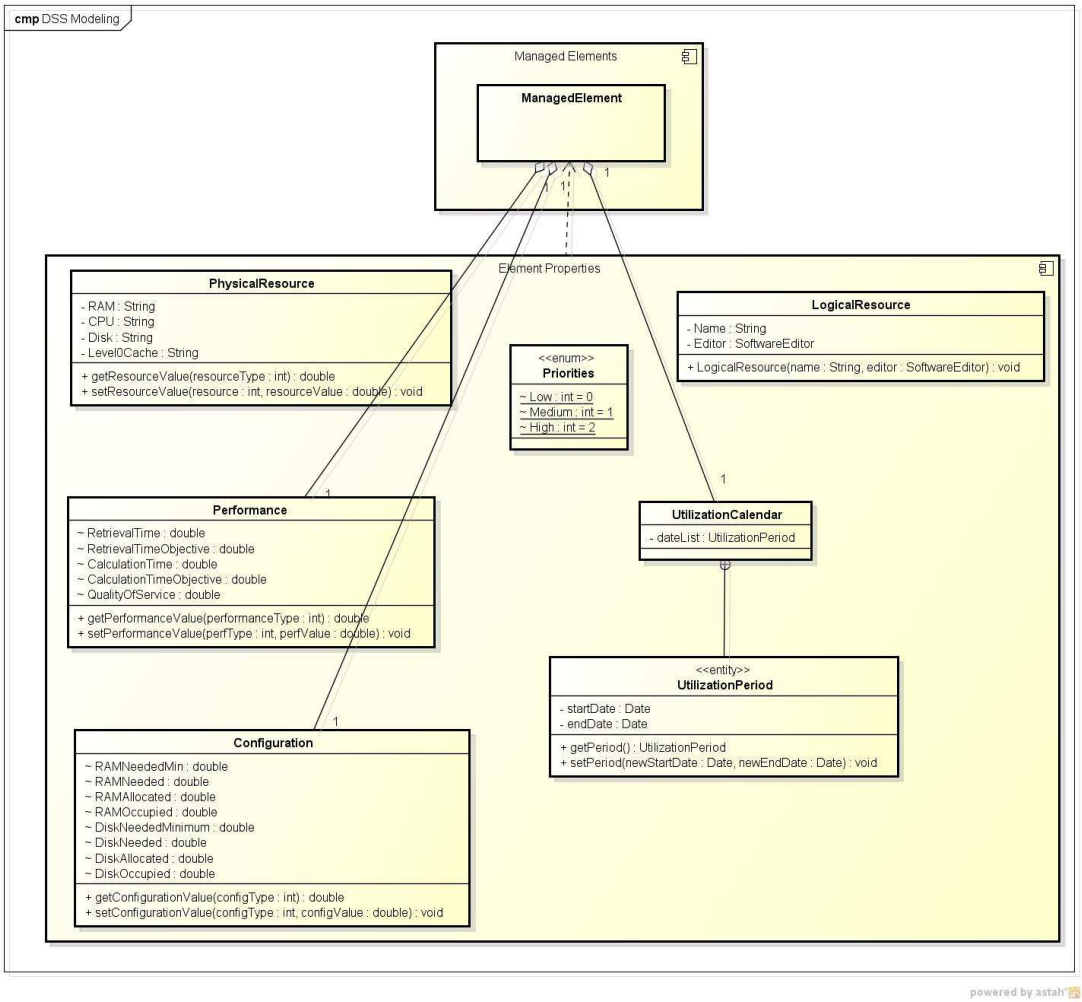


Figure 52 : Managed Elements Properties

7.2.1.3.1 Resources

Resources are used by the managed elements to assure their functioning. There are two types of resources: physical resources (e.g. hard disks, RAM memory) and logical resources (e.g. operating systems, software products).

Physical Resources – A physical resource assures the functioning of the managed element from the hardware point of view. The term can be misleading, as physical may also refer to the human experts. In our model, a physical resource is strictly related to the hardware aspect. It is a part of the physical machine (the physical server) and includes the hardware elements used further by the managed elements. These include:

- *RAM (String)* – the code of Random Access Memory installed on the physical machine. This RAM is used further by the logical servers and for base cache allocations. A RAM memory can have several specifications such as its type (DDR2, DDR3, SDR), its sizes (M,S), its capacity (2,4,16) etc. All these are contained in its fabrication code. A physical machine can have several RAM bars.

- *CPU (String)* – the code of processor installed on the physical machine, which contains several specifications such as platform (32,64 bit), clock speed (2,6Ghz), level cache sizes (level0:128Ko,level1:2Mo) etc. A physical machine may have several CPUs as part of its architecture.
- *Hard Disk (String)* – the identifier of the hard drive, with reference to its capacity (1To), its connection type (SATA, SATA2, IDE), format size (2,5”; 3,5”), I/O performances (w30Mo/s;r100Mo/s) etc. As a physical machine usually contains several hard disks, the identifier could also specify the architecture in which the disks are connected (Independent, RAID0, RAID1).

Logical Resources – a logical resource assures the functioning of the managed element from the software point of view. Examples of logical resources include operating systems or software products used with the management of the DSS. Usually the logical resources are linked (as see in the overall schema) with a software editor (from the Software Editor class). Logical resources make use of physical resources to assure their functioning. For example a Windows Server 2008 64bit can only be installed on a 64bit processor. A logical resource has only two properties, alongside the user relation with the Physical Resources: its name and its editor.

7.2.1.3.2 Indicators

The Indicators group contains (i) the performance and measurement analytical indicators that are either aggregated or computed and separated into different classes in function of their purpose, and, (ii) several enumerations classes that include the constant elements used by the managed elements to describe either their functioning or their performance. We describe each of them, starting with the enumerations.

Priorities – enumeration of the types of priorities a managed element has. A priority influences the amount of physical resources allocated to the element and its performance metrics. For example, a low priority base receives 3 times less RAM memory to use for its caches than a high priority base. There are three types of priority:

- *High* – the highest priority of a managed element.
- *Medium* – the intermediate priority, where scales are defined such that managed elements receive less physical resources than the high priority ones.
- *Low* – the lowest priority, with the biggest scales, such managed elements receive the smallest amount of physical resources and have the lowest performance objectives.

Program Types – the enumeration contains the possible types. The type of the program determinates the corresponding performance indicator. For example, a reporting program generates the base query response times, a calculus program generates the calculation times. The types of program are:

- *Report* – programs that implement reporting scripts, which generate values for the query response times when retrieving data from the bases.

- *Calculus* – programs that implement calculation scripts, which generate the calculation times for base calculation operations.
- *Restructuration* – programs that use restructuration and dimension changing, generating the restructuration indicators. Examples of restructuration are: changing the number of dimensions to a base, adding new dimensions and aggregation points, reloading the data in the bases.
- *Export/Load* – the programs that are in charge of exporting/loading base data.

Utilization Calendar / Periods – the class contains the means of representing the utilization periods of the managed elements. There are two cases, either a managed element is in its utilization period (meaning it is used) or either it is not. This sort of policy needs to be described in order to decide how physical resources are allocated and how performance is measured depending on each period. The level of detail of the utilization periods is the day, based on the day/night usage purposes. Each managed element can have one or more utilization periods, and each utilization period can correspond to one or more managed elements. The properties used to describe a period are:

- *ID (String)* – the unique identifier of the utilization period
- *StartPeriod (Date)* –the starting day of the utilization period
- *EndPeriod (Date)* –the ending day of the utilization period

Configuration – the configuration class contains the configuration indicators. These are usually related to the allocation of physical resources to the managed elements. Configuration parameters aggregate as well over the DSS architecture. For example, a base requirement for RAM memory is transmitted as a RAM requirement for the physical application, which in turn is transmitted to the logical servers and finally to the physical server.

- *RAMNeededMinimum (double) (Mo)* – the amount of RAM memory needed by the element to function at its minimum
- *RAMNeeded (double) (Mo)* – the amount of needed RAM memory specified by the element for the best possible performances
- *RAMAllocated (double) (Mo)* – the amount of allocated RAM memory to the element
- *RAMOccupied (double) (Mo)* – the amount of RAM memory that the element is currently using. It is not always equal as the allocated RAM.
- *DiskNeededMinimum (double) (Mo)* – the amount of disk space required by the element to function at its minimum
- *DiskNeeded (double) (Mo)* – the amount of needed disk specified by the respective element for the best possible performances
- *DiskAllocated (double) (Mo)* – the amount of allocated disk space to the respective element
- *DiskOccupied (double) (Mo)* – the amount of disk space that the element is currently using. It is not always the same as the allocated disk.

Performance – refers to the indicators that assure the measurement of performances of the managed elements. The performance indicators aggregate over the managed elements hierarchy.

- *QRT (double) (s)* – the query response time measured in seconds, obtained after data retrieval operations.
- *QRTObjective (double) (s)* – the objective query response time for the respective managed element. It depends on the importance, priority and utilization period of the managed element, in conformity with the SLA agreements.
- *CalculationTime (double) (s)* – the time obtained by calculation operations on the bases, via calculus programs.
- *CalculationTimeObjective (double) (s)* – the objective given for the calculation time, by following the same consideration as the QRT objective
- *QualityOfService (double) – user satisfaction.* As mentioned earlier, the quality of service expresses how far the actual performance level from the objective performance level is (as a rapport).

7.2.2 Autonomic modeling

The last part of the UML modeling focuses on the autonomic system adoption. It is based on the DSS model and adds the elements that allow the implementation of the Autonomic Computing specifications for self-X behaviors (Figure 53). We notice that the connection is done via the Managed Element, each managed element disposing of one or more autonomic computing managers.

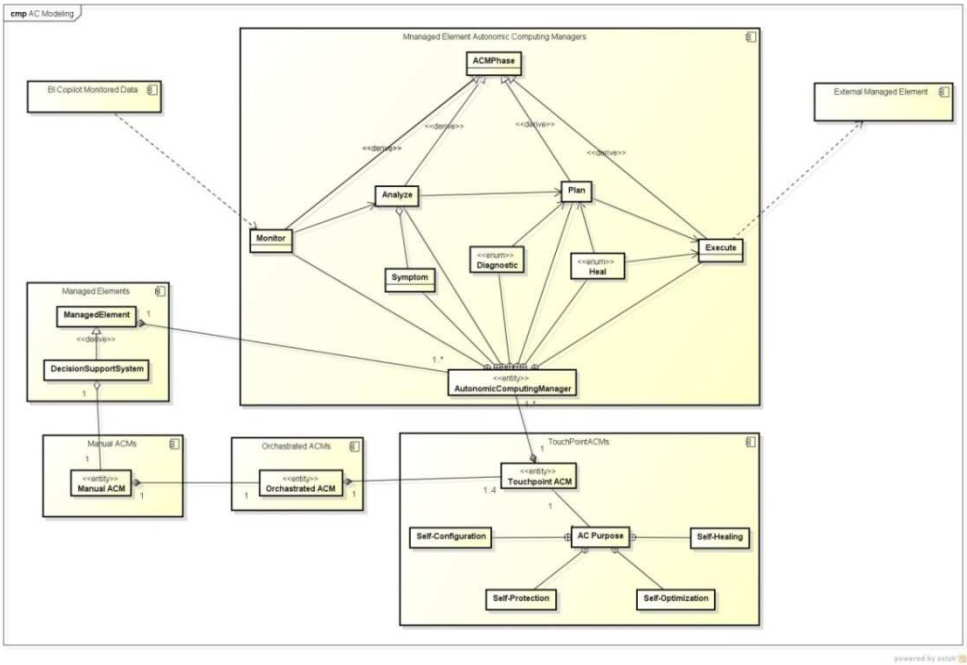


Figure 53 : Autonomic Computing Adoption Diagram

The elements of the diagram are described next:

Managed Element – the link with the previous modeled managed element. This is also known as the link towards the *Static Knowledge Base.* We make this separation between static and dynamic knowledge, as the dynamic knowledge is represented by the rules describing the autonomic computing phases. The ontology model will explicitly present the two.

Autonomic Computing Purpose – represents the four principles of an autonomic adoption system (C.H.O.P. principles). Each of the principals is linked to one or more autonomic computing managers.

Element Autonomic Computing Manager – the modeling of the manager corresponding to the managed elements from the DSS model. Each managed element from the DSS hierarchy is bound to one or more autonomic managers. However, there can be only one manager per purpose (i.e. we can't have two self-optimization ACMs for the same base). The behavior of each manager is specific to the hierarchy level of the managed elements (e.g. for all bases, the manager behaves the same, even if different instances are created). From the diagram, we can distinguish the MAPE-K loop around a rule knowledge base (linked via 'use' relations). This rule knowledge base represents what we call the dynamic knowledge.

Autonomic Computing Manager Phases - which include the *Dynamic Knowledge.* It contains the rules for the manager's phase passage (from monitor to analyze to planning to execute) along with the intermediate created states (diagnostics, symptoms and heals). Moreover, the dynamic knowledge contains information related to the algorithms/heuristics used by the MAPE-K loops.

Touchpoint Autonomic Computing Manager - The touchpoint manager is used for regrouping several element autonomic managers that share the self purpose. It is an aggregated manager, as it provides the first overview over the activity of the element managers from the system. It also allows distinguishing the levels of autonomic adoption from the implementation levels of the self-X factors.

Orchestrated Autonomic Computing Manager – The orchestrated manager is used for regrouping several touchpoint managers depending of the chosen autonomic implementation. It is sub-related to the manual manager (a manual manager can have several orchestrated managers). The orchestrated manager requires the static knowledge (via a 'use' relation) to function, as well as specific commands given by the expert from the manual manager.

Manual Autonomic Computing Manager –the manual manager is connected to the static knowledge as it makes use of this information (via a 'use' relation). We notice its relation with the overall Decision Support System managed element and the interface for its management.

Note There are two elements that have not been introduced in the model: the sensors and the effectors.

- *The sensors* - represent the entry point information that is used by our system. On the diagram it is shown as the BI Copilot monitored data. This information is assured by the software module developed by tSP2 and is only mentioned as a used component in the works carried with this thesis.
- *The effectors* - The same considerations are given for the effectors, which in our system are implemented directly either via rules or via third party software used by the AC adoption model. As these elements are external they have not been introduced in the internal description diagrams. They are represented on the general schema as the External Managed Element.

7.2.2.1 Managed element autonomic computing manager

The Element Autonomic Computing Manager represents our modeling proposition in rapport with the autonomic manager from IBM's hierarchy over the lowest levels (the intelligent loop over the managed resources). It is shown in Figure 54.

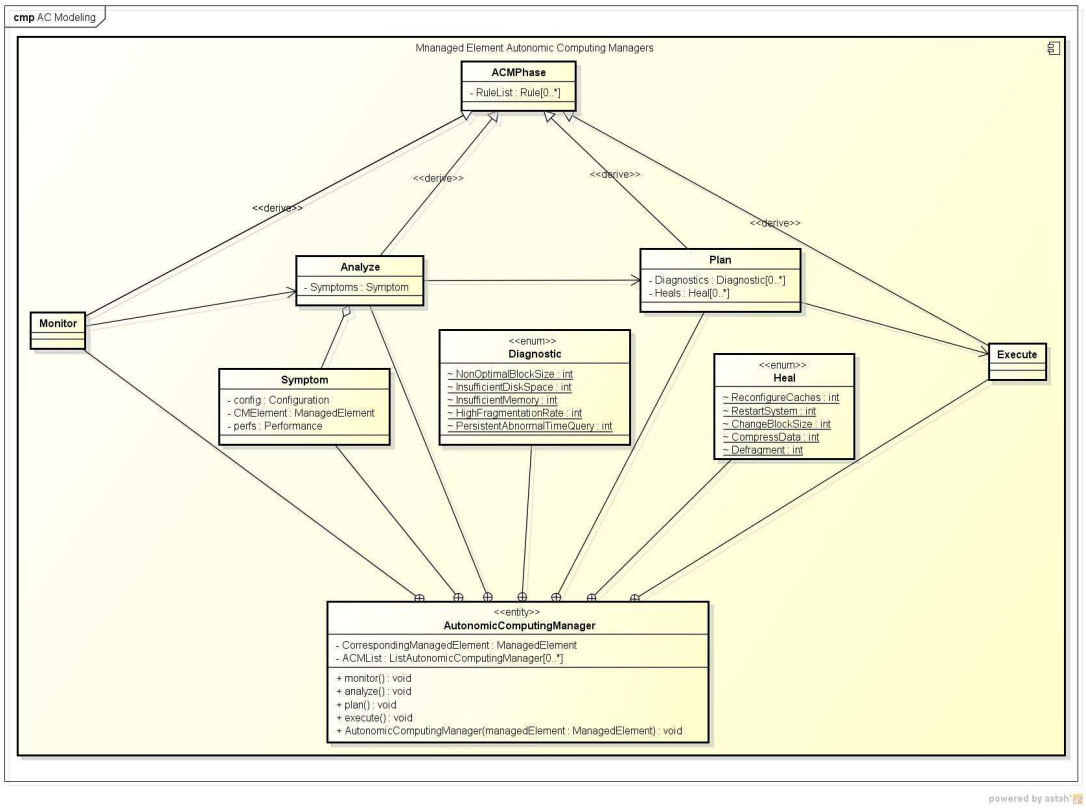


Figure 54 : The Managed Element Autonomic Computing Manager

The element manager is defined in concordance with its associated managed DSS element. Still, we would expect that a certain managed element has the same behaviors, or that the behavior policies are the same. For example, if we are faced with an instance of the Base class, that is a base; its behavior patterns are already defined for its type. Therefore, autonomic managers' behaviors are distinct by DSS hierarchy levels.

Autonomic Computing Manager – the element autonomic computing manager, linked to a single managed element and, eventually, to other element managers. It has the following properties and methods:

- *Corresponding Managed Element (Managed Element)* – the element that the ACM manages
- *ACM List (list: Autonomic Computing Manager)* – the list of element ACMs with which the manager communicates. For example, an application ACM would have the list of the base ACMs.
- *Monitor()* – the function that implements the description of the monitored parameters and the associated rules.
- *Analyze()* - the function that implements the operation of analysis of the monitored elements and the associated rules.
- *Plan()* **–** the function that implements the operation of planning the required actions from the analysis process.
- *Execute()* – the function that implements the execution process. This can be distributed, as actions don't always refer to the same targets (i.e. adding an extra RAM bar and afterwards reallocating the cache memory).

ACM Phase – the generic class that describes the four phases of the MAPE-K loop. A phase is described by an entailment of several rules, that share the same loop step (monitor, analyze etc.). An ACM phase contains a single parameter:

- *List Rule (Rule)* – the list of rules that corresponds to this phase. The order in which rules act is the other in which they are added to the list (meaning that the rules are ordered by their execution priority). If two rules have the same priority, the order in which they are stored in the list doesn't matter.

Monitor – represents the extension of the ACM Phase for implementing the monitoring capabilities. The objective of the monitor phase is to provide the elements needed for the analysis, the symptoms.

Symptoms – are the results of the monitoring phase, and include the characteristics that describe the managed element. The synonyms have the following properties:

- *Config (Configuration)* – an instance of the configuration, containing the configuration indicators.
- *Perfs (Performance)* – an instance of the performance, containing the performance indicators that are monitored.
- *CM Element (Managed Element)* – the managed element which has the symptoms. An instance of symptoms applies for one managed element and one managed element only.

Analyze – represents the extension of the ACM Phase for implementing the analysis capabilities of the ACM. Taking the symptoms as entry point, the analyze phase obtains a series of diagnostic based on the analysis rules.

Diagnostics - are the result of the analyze phase, and apply at general level or for a specific managed element. There are three levels of diagnostics:

- *Red (Error)* – indicating a problem that requires immediate attention and potentially can cause the system to stop working (e.g. the base index cache value is lower than the minimum accepted value, thus causing the respective base to no longer work).
- *Orange (Warn)* **–** meaning that there is a potential upcoming error. The given elements are not critical enough for give a red diagnostic, but they could become critical in the future (e.g. the base index cache is constantly lowered and gets too close the minimal accepted value).
- *Green (OK)* **–** expressing that there is nothing wrong with the analyzed symptoms.

The diagnostics class is an enumeration that contains the list of possible diagnostics, organized by the three levels of importance (error, warn, OK), such as:

- *NonOptimalBlockSize* – indicating that the values for the block size of a base are not optimal (warn)
- *InsufficeintMemory* – the RAM memory required on the physical server is not sufficient (warn and possibly error)
- *InsufficientDiskSpace* – the disk space needed for the storage of the bases is insufficient (error)
- *HighFragmentatoniRate* – indicating that the levels of fragmentation of the base files on disk are too high (warn)
- *PersistentAbnormalTimeQuery* – the queries take too long to respond (in rapport with given expectations) and this state is persistent (warn or possible error).

***Plan* –** represents the extension of the ACM Phase for implementing the planning capabilities of the ACM. The planning phase contains the order in which the potential changes required execute (here the healing actions). Its parameters are:

- *Diagnostic (list:Diagnostics)* - the list of diagnostics for the concerned analyze indicators.
- *Heal (list:Heals)* **–** the possible heals that correspond to the bad diagnostics

***Heals* –** represent a list of the possible actions that can be taken in order resolve the bad (orange and red) diagnostics. These actions vary depending on the associated C.H.O.P. principle (e.g. change an index cache value for better performances (self-optimization) or change an index cache value because its current value is under the minimum specified threshold (self-healing)). The actions are linked to diagnostics, as a single diagnostic may have several heals, and several diagnostics may have the same heal. Some examples of heals are:

- *ReconfigureCaches***–** the action indicating that caches should be reconfigured (corresponding to the *InsufficeintMemory* diagnostic***)***
- *ChangeBlockSize* – the action of a change in the block size configuration parameter (*NonOptimalBlockSize****)***

- *ComrepssData* – that indicates a compression of the files on disk (*InsufficientDiskSpace**)***
- *Defragment* **–** indicating a defragmentation is required (*HighFragmentatoniRate**)***
- *RestartSystem* **–** an OS restart will be performed to validate changes (*InsufficientDiskSpace, InsufficientDiskSpace**)***

Execute – represents the extension of the ACM Phase for implementing the execution capabilities of the ACM. This is a little different from the previous phases, as an execution action may or may not be performed in an autonomic way (e.g. adding a new RAM bar requires human intervention, while changing the value of a cache parameter can be done in an autonomic manner).

We have seen that an ACM corresponds to a single managed element and a single managed element only. From the DSS modeling, the managed elements are organized hierarchically, thus the managed elements ACMs are in turn organized the same way. This is important as ACMs need communication between them due to common elements, such as monitored configuration / performance indicators. The ACMs hierarchical organization is shown in Figure 55.

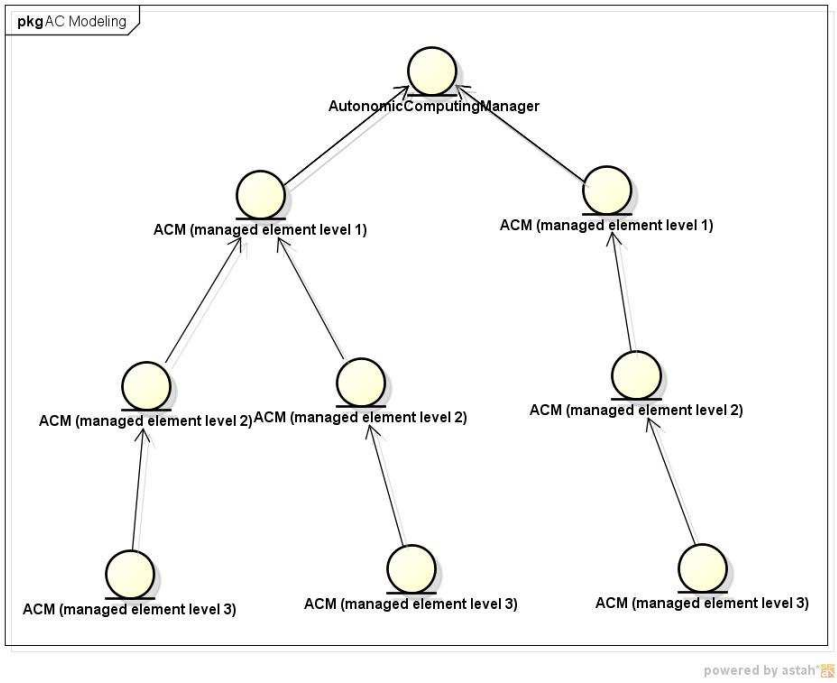


Figure 55 : Logical organization for the managed elements levels ACM

7.2.2.2 Touchpoint autonomic computing manager

Responsible for the implementation of the self-X autonomic adoption characteristics, the touchpoint manager aggregates the element autonomic computing managers. The aggregation is over the purpose of the element's managers. The implementation of the self-X functions remains at the choice of the system architect, based on the needs and various constraints (such as cost).

For instance, one may choose to implement only the self-optimization aspect, where all the element autonomic managers focus on optimizing the system performance. If we consider a base autonomic manager for self-optimization, its sole purpose would be to improve the specified performance indicator (e.g. query response time). Any other considerations, such as base security access or data corruption, are therefore ignored by the self-optimization manager. These considerations would be treated by the self-healing and / or self-protecting touchpoint managers. Nevertheless, the dependency between the four aspects supposes that for the self-optimization implementation, self configuration and self optimization have already been assured prior to it. Figure 56 presents the touchpoint computing manager:

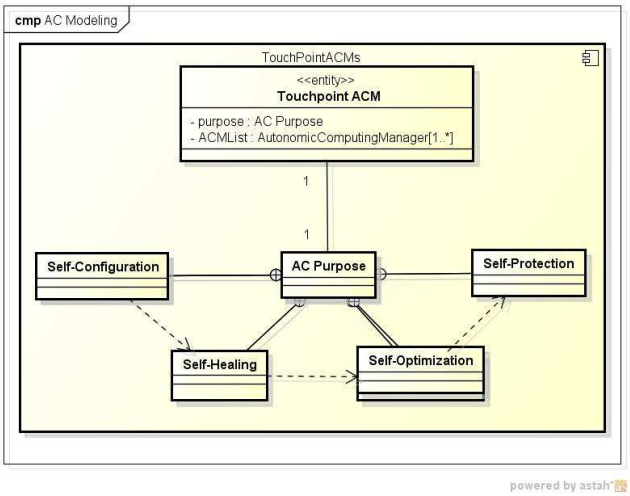


Figure 56 : Touchpoint Autonomic Manager

Touchpoint ACM – describes the manager that is in charge of assuring that the C.H.O.P. principles are applied. As we can see, there is a 1 to 1 relation between the manager and the autonomic purposes. Each purpose corresponds to one touchpoint manager, and one only. It has the following properties:

- *ACM List (list:Autonomic Computing Manager)* – the list of managed element ACMs that share the same purpose. The list must have at least one managed element ACM. If no element is present, then the respective C.H.O.P principle isn't implemented by the autonomic system.
- *Purpose (list:AC Purpose)* – the purpose that is assured by the touchpoint ACM.

AC Purpose – represents the four autonomic computing purposes (the C.H.O.P. principles). The modeling of the purposes also includes the dependence between the four principles:

- *Self-Configuration* - the first principle, which assures the configuration of the DSS. It can be seen as the level 0 of autonomic adoption.
- *Self-Healing* – the second principle, which assures the diagnostics and the repairing actions (if necessary). Self-Healing is dependent on the previous principle, as it

requires a configuration in order to realize if something is wrong or not. Can be seen as level 1 autonomic adoption.

- *Self-Optimization* – level 2 of autonomic adoption, which assumes the existing of a configured and well working DSS. Once the first two principles are assured, the DSS can proceed further to try to optimize its functioning.
- *Self-Protection* – the last level of autonomic adoption (level 3), implies the existence of a well configured, well working and eventually optimized DSS before assuring its protection. As the DSS is now in its 'best' possible from, it must be protected from interference with its functioning state.

7.2.2.3 Orchestrated autonomic computing manager

The orchestrated manager is in charge of regrouping the existent touchpoint managers. The initial (IBMs) description of the orchestrated manager is by regrouping the touchpoint managers by discipline. In our case, we consider the existence of a single orchestrated ACM, modeled on top and in charge of the entire DSS (Figure 57).

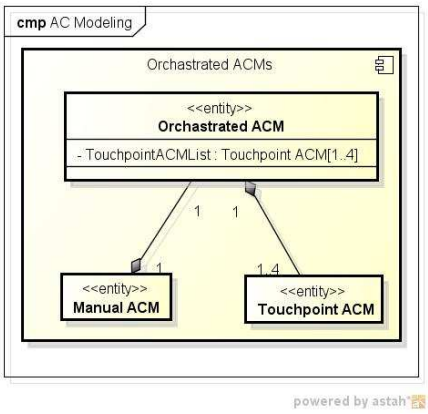


Figure 57 : The orchestrated autonomic computing manager

Orchestrated ACM : aggregates the (maximum) four touchpoint managers. There is a 1 to 4 relation between the orchestrated and the touchpoint managers, and a 1 to 1 relation with the manual manager. It has only one property, which is the list of associated touchpoint ACMs:

- *Touchpoint ACM List (list:Touchpoint Autonomic Computing Manager)* – the list of touchpoint ACMs. The list must have at least one touchpoint ACM. If no element is present, then the respective C.H.O.P principle isn't implemented by the autonomic system.

7.2.2.4 Manual manager

The last component of the autonomic computing adoption scheme is the manual manager. The role of the manual manager is to offer the interface between the system expert and the functioning of the autonomic system. As the role of the autonomic adoption is to

assist the expert, with the help of the manual manager the expert will be theoretically able to override any decision or action of the autonomic system.

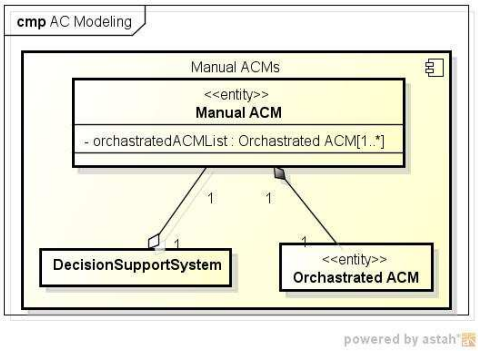


Figure 58 : Manual Manager

Manual ACM – Firstly, it has a 1 to 1 link with the DSS, as each DSS has a manual manager. It is the connection point to the rest of the autonomic architecture. Secondly, another link is made with the Orchestrated ACMs, as the manual manager manages the orchestrated and further on touchpoint managers (depending on the system's implemented principles).

7.2.3 Knowledge base model - dynamic

This section describes the last part of the model, by links the DSS modeling and the AC adoption modeling. The DSS model has shown our proposition over the physical and logical DSS architecture. The AC adoption model presented further the proposition of integrating autonomic computing over the proposed DSS model elements. This last part, the knowledge base modeling can, is seen as the continuity of the AC adoption modeling as it shows in details how we have conceived the information in the core of the AC adoptions.

We have called this 'dynamic', as it models the information that is likely to change over time and more important that changes in function of the existing environment. 'Dynamic' is also aimed at describing how information fits in the MAPE-K loops, the loop phase passage and how the different ACMs communicate.

The general overview of the dynamic modeling is shown in Figure 59. There are three elements that form up the dynamic knowledge base: the Knowledge Source, the Rules and the Heuristics. The schema puts them in evidence while interconnecting with the already presented elements.

Knowledge Source – represents the information sources that provide knowledge for the dynamic knowledge base (rule formalization). The knowledge source can be found under different forms: readme documents, technical forums, the human experience etc.

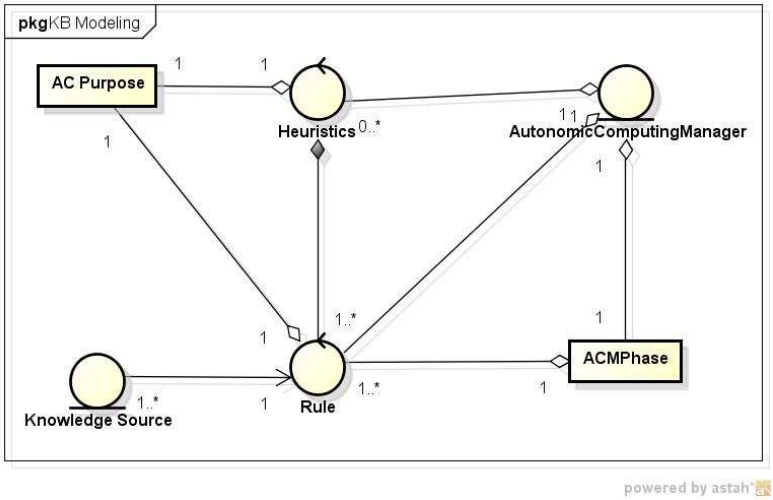


Figure 59 : The Dynamic Knowledge Base UML component diagram

Rule – represents the formalization of all the ECA (event condition action) information. The rules can be seen as the 'fuel' for the autonomic computing manager engine. They are differentiated by several factors such as: priority, type or action. The organization is discussed in detail in the next subsections. There is a 1 to 1 link with the AC purpose, as each rule is part of a group expressing one of the four purposes. An ACM contains one or several rules. This ACM link is reinforced by the link with the phases of the ACM as each rule corresponds to one of the four steps of the MAPE-K loop, thus each step includes a certain number of rules. As we speak here of knowledge formalization, there are two phases in defining each rule: (i) the formalization and (ii) its integration/impact with the existent rules.

Heuristics **–** represent the second part of the dynamic aspect and formalize the algorithms/heuristics used for the self-configuration and self-optimization implementation. A heuristic uses one or more formalized rules. Moreover, the heuristics are purpose specific and they are implemented by an autonomic manager. There are ACMs designed to implement several heuristics, whereas others implement no heuristic at all.

Before going into detail with each of the three elements above, a description of the autonomic adoption dynamics is shown, to argument the introduction of the dynamic knowledge and its functioning with the autonomic system.

7.2.3.1 Autonomic computing adoption model - dynamics

The AC adoption dynamics refers to usage of the dynamic knowledge rules with the ACM loops. There are several criteria based on: the autonomic principle (CHOP), the autonomic manager phases (MAPE-K) and the place of the managed element in the DSS hierarchy. We present below two use cases: first a general use case, and then a detail of the Self Healing principle.

7.2.3.1.1 ACM inter-communication

The general use case describes the dynamics of the rules over the four CHOP principles, with regards to their priority (Figure 60).

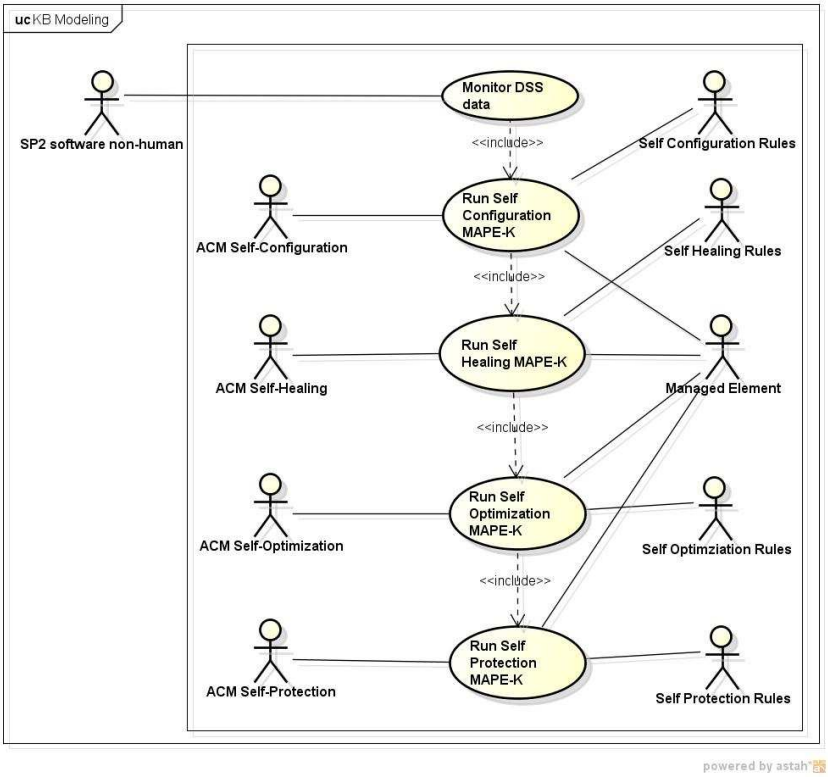


Figure 60 : AC CHOP principles dynamics

- On top, we rediscover the SP2 software solutions that monitor the data concerning the DSS.
- Then we have the level 0 principle adoption with the self configuration MAPE-K loop. The loop is performed by the managed element ACMs that correspond to the Self Configuration touchpoint ACM. It is based on the set of Self Configuration Rules.
- The Self Healing MAPE-K loop follows next, as the system is now configured. It is based on the Self Healing ACMs and the Self Healing Rules
- The Self Optimization MAPE-K loop which will execute once the DSS is functioning and OK from the point of view of Self-Healing.
- Last, once the DSS is well configured and optimized, with no problems, we find the Self-Protection loop, with its associated ACMs and Rules.

The communication between the managers is done by managed element, by purpose, by ACM loop, by level of autonomic manager. So there are four criteria in the mentioned order. IBMs specifications present only a communication between the phases of the ACM loop and the levels of autonomic managers. No other indications are given, thus our approach introduces the two other criteria.

Communication by managed element hierarchy introduces a communication order between the ACMs in the lowest level of the adoption model, via several links between them. Moreover, some of these links express the hierarchy of elements. The communication between the ACMs of the managed elements is done on the oriented hierarchy graph, in a bottom-up manner. For an ACM to communicate with an identical or lower level ACM it must pass through the common upper ACM (from the managed element hierarchy point of view).

We make a clear distinction between the specific information that is used to manage a managed element only, and the shared information used by the upper/lower levels. The specific individual information is treated in the MAPE-K loop separately in a continuous manner (no loop interruption). On the other hand, the shared information will be used through the ACM loops and requires synchronization. The passage of information is done either top-down or bottom-up, depending on the loop phase:

- *Monitor*: the order of the operation is not important; the purpose being to obtain all data on the DSS. As data is recovered from log files, the used approach is top-bottom, but again we have no guidelines on this. We leave it at the choice of the implementation, as the process can even be parallelized.
- *Analyze*: as this is performed based on aggregated information, we use a bottom up approach to the execution of the phase. For example an application analyze phase would depend on the results of the base analyze phase.
- *Plan:* planning is based on element dependence (i.e. we cannot stop and reconfigure the parameters of a base as long as the application turns). Therefore, this phase is done in a top-down manner, starting from the DSS and ending with the bases.
- *Execution:* based on the same dependence as the planning, the execution will be done in a bottom up manner (i.e. we can't change the configuration of the base once the application is restarted).

Communication by purpose - IBM gives a 'hint' by presenting the touchpoint managers in the CHOP order, but puts no link between the four. We introduce the purpose dependence of each managed element ACM by a functional relation. We propose the usage of *not one* ACM by managed element *but four*, one for each purpose. The communication is done in order of the four principles. For example, we want to assure self optimization. Starting the self optimization ACM would trigger (and wait) the self healing ACM which in turn triggers and waits for the self configuration loop to finish. Table 21 below shows this dependence.

Each individual MAPE-K loop will be fully executed before triggering the next loop. For instance, the execution phase of the self configuration ACM will have as a last command the trigger towards the monitor phase of the self healing ACM. The functioning manner is iterative, thus synchronization is assured. This separation assures an easier access to the specific level of autonomic adoption depending on the needs and objectives.

Table 21 : Self Optimization adoption ACM communication

ACM	Phase	Action
Self Configuration	Monitor	
	Analyze	
	Plan	
	Execute	last: Trigger Self Healing ACM monitor phase
Self Healing	Monitor	
	Analyze	
	Plan	
	Execute	last: Trigger Self Optimization ACM monitor phase
Self Optimization	Monitor	
	Analyze	
	Plan	
	Execute	

7.2.3.1.2 Specific ACM communication – self healing

As a detail of the general use case, this subsection presents the Self Healing dynamic diagram. This is valid for the other three principles also. We have chosen self healing as it is easier to understand intuitively. It is shown in Figure 61.

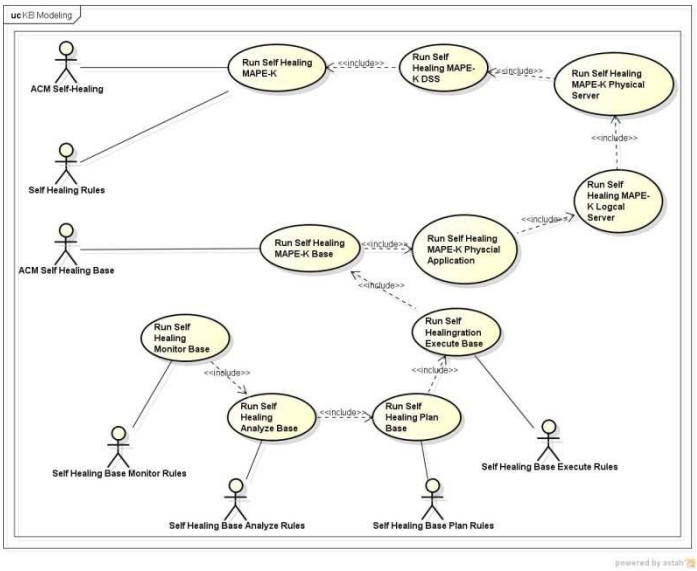


Figure 61 : Self Healing detailed dynamics

The Self Healing general loop implies running the loops for each of the managed element, in the ordered specified by the DSS hierarchy: first the self healing base loop, then the physical application, then the logical server, and so on.

Each of the self healing managed element loops are then detailed over the four phases of the ACM, by running the specific rules for monitoring, analysis, planning and execute. In this logic, the dynamics of the self healing principle is: *base monitor->base analyze->base plan->base execute->physical application monitor -> physical application analyze -> physical application plan -> physical application execute -> logical server monitor -> logical server analyze -> logical server plan -> logical server execute -> physical server*

monitor -> physical server analyze -> physical server plan -> physical server execute -> DSS monitor -> DSS analyze -> DSS plan -> DSS execute.

This permits a clear organization of the rules, over the criteria of each AC principle, ACM phase, managed element, offering a better control over the entire system. The static knowledge is common; whereas the dynamic knowledge is organized individually according to the presented criteria.

7.2.3.2 ACM knowledge source

The knowledge sources contain the semi and unstructured information that is used for rule formalization. Figure 62 shows the diagram of the knowledge sources. There are three categories, based on our experience with working with DSSs.

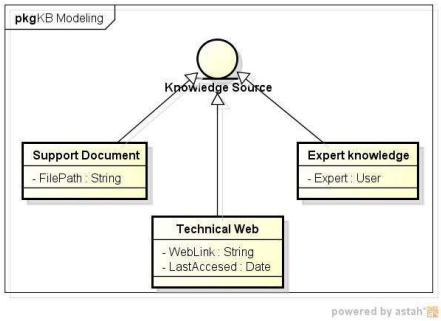


Figure 62 : Knowledge Source detail diagram

Knowledge Source – is the base entity that describes a source of information. It has the parameters.

Support Document – represents any document provided by a software editor, in any format (.doc, .pdf, .rtf etc.). It has a single parameter:

- *FilePath (String)* – the path of the file, pointing towards a disk location or a web address from where the file can be downloaded

Technical Web – represents any knowledge source that is found on the web, generally technical forums or online product documentations. A technical web knowledge source has two parameters:

- *WebLink (String)* – the web link of the source
- *LastAccesed (Date)* **–** the date at which the source has been last accessed. There are many cases in which online resources are no longer available after a certain amount of time. This is why, where possible, we always try to migrate a technical web source into a support document, so that the information remains persistent.

Expert Knowledge – represents any information that comes from a human expert. As it is the experts who are in charge of rule integration, we can go as further as to consider that ideally all readme documents and technical web sources are integrated in the expert's knowledge. Still for the modeling, we have chosen to separate them. At most, we can find

the same piece of information linked to different sources (e.g. readme documents and expert). There is only one parameter for the expert knowledge:

- *Expert (User)* – a link with an instance of the User class, who has the status of DSS expert.

7.2.3.3 Rule

This section details the modeling of the Rule concept. As mentioned earlier, rules are organized depending on several criteria and serve to different purposes. Still, their common central point is the autonomic manager and its knowledge base with which the rules are integrated. Consider the following example for a rule:

Example. The following advice is expressed in an Essbase readme document: The available RAM memory should be distributed between the index cache and the data and data file cache in a manner that the data cache is at about 12,5% of the data file cache, for the BSO bases. (i) The first step is the formalization of the rule with the elements described in the system. We identify the configuration parameters: the available server RAM memory and the index, data and data file cache parameters for a base. With these parameters the rule is formalized with the chosen technical solution. (ii) Once the rule formalized an impact analysis with the already existing rules in the system is performed. If there are any inconsistencies or ambiguities a human expert intervention is required for its integration (e.g. another rule states that the data cache should be at 20% of the data file cache).

The details of the rule concept are shown in Figure 63.

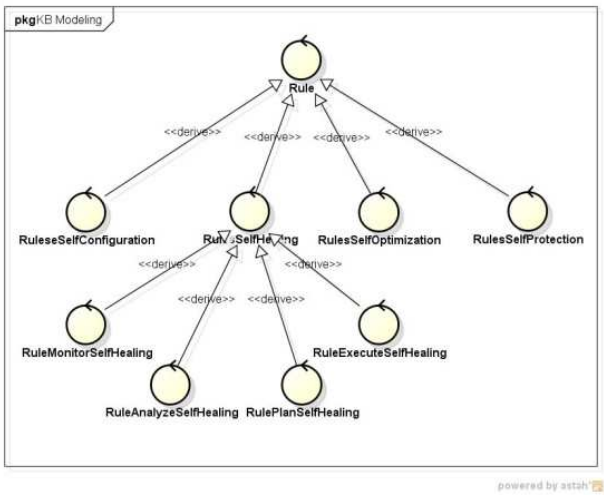


Figure 63 : Rule Organization Diagram

Starting from the general concept of rule, a first division is made in function of the AC principle. Each of these sets is in turn divided according to the MAPE-K loop phases. The diagram details this for the Self Healing principle (the others have the same organization). The attributes that characterize a rule are:

- *Name (String)* – the name of the rule. We propose a series of prefixes specific such as to determine to where the rule belongs: *ACMManagedElementType _ CHOPPrinciple _ MAPEKLoopPhase _ RuleName (e.g. Base _ SelfOptimization _ Analyze _ Rule1).*
- *Body (String)* – the body of the rule, its presentation in the implementation formalism (i.e. SWRL, Jena, SPARQL etc.).
- *RuleSource (KnowledgeSource)* – the source from which the rule has been formalized. As mentioned earlier, a rule can have several sources.
- *Purpose (AC Purpose)* – the CHOP principle purpose for which the rules applies. In the general schema, a rule corresponds to one purpose and one purpose only. A situation may occur with monitoring rules, where one rule may apply for several purposes. In this case, the rule is duplicated for each purpose and referenced differently in function of its purpose.
- *ACM (AutonomicComputingManager)* **–** the ACM and implicitly the DSS managed element for which the rule applies. We keep the same consideration for rule duplicity as in the purpose case. A rule will be duplicated for different ACMs.
- *ACM MAPE-K Phase (ACMPhase)* **–** the phase of the ACM loop for which the rule applies.

7.2.3.4 Heuristics

The heuristics are placed in the dynamic knowledge schema as a rule usage entity. They make use of the corresponding rules integrated with the ACM phases. We present further in this section our modeling for heuristic integration, by showing two examples for self-optimization and self-configuration.

The role of the heuristics is to find better and/or optimum configurations given (i) a starting point and (ii) the conditions which determine if a configuration is good or not. We specify that the objective is not to find the best possible configuration; rather, it is to find a good configuration that assures the required level of performance. Figure 64 shows the detailed heuristics.

Heuristics – the generalization heuristics entity. A heuristic is adopted by an ACM. Even if a heuristic corresponds to a DSS management element, this link is implicit by the ACM that manages the DSS element. Any heuristic has the following properties:

- *Name (String)* – the generic name of the heuristic. We model the names so that it reflect the managed element and the purpose of the heuristic (e.g. Base_SelfConfiguration_Heuristic1).
- *Rules (list:Rule)* **–** the list of rules that concern the heuristic implementation.
- *ACM (AutonomicComputingManager)* **–** the ACM which implements the heuristic. As a heuristic contains a set of rules, the requirement is that these rules are all part of the same ACM.

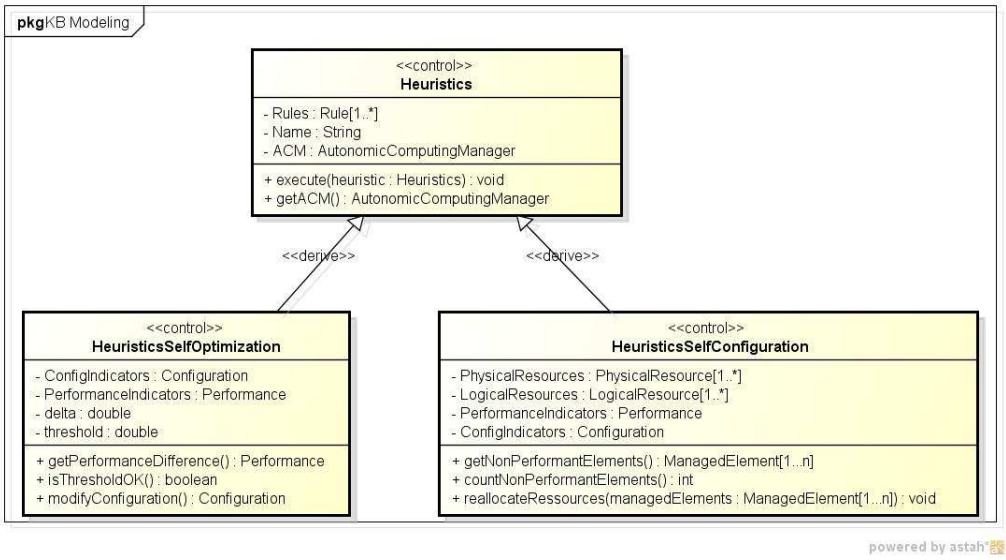


Figure 64 : Detailed Heuristics Diagram

There are two extensions of the heuristics corresponding to the two CHOP principles that we use for exemplification: Self Configuration and Self Optimization. The two are detailed in the subsections below.

7.2.3.4.1 Self optimization heuristics

The objective of the self optimization heuristic is to assure that the DSS performance indicators are at good levels while using the least of resources. The principle is: *acceptable performance with fewest possible resources*. With common data warehouse configurations, resource allocation isn't done intelligently according to the performance levels. Consider the following example.

Example. a client requirement: the data warehouse average query response time should not be greater than 2s. For the simplicity of the example we consider the QRT linked only to the quantity of RAM memory allocated to the DW. With the current configuration of 16GB of RAM, the obtained average time never crosses the value of 1.5s. By performing a series of repetitive tests, an expert concludes that 12GB of RAM suffice for assuring the required threshold of 2s. At each test, the expert decreases the amount of allocated memory, and performs a series of queries to see how the new average QRT is influenced.

What we propose with this heuristic is to perform the series of tests (adjustments) done by the expert, with a higher granularity. This requires several recurrent steps before reaching the optimal point resources / performances. This point depends on the number of iterations as there will always be an error. There are two specific running parameters of the self optimization heuristic:

Step - the notion of step or passage is used to determine the frequency of the changes. Based on the DSS Usage Purpose, a step corresponds to a period of one day. As we recall at the end of operational day, the new data is aggregated, thus the impact of the configuration parameter change is analyzed and new configurations are submitted for testing. This

corresponds to one MAPE-K loop passage. Alternatively, the step corresponds to a predefined period at the end of which statistics are gathered (i.e. once each 5 minutes, once each 3 days etc.).

Activation Periods – based on the DSS Utilization Periods, the activation periods of the heuristics depend on when the targeted managed element is used. More specifically, as the optimization operation influences directly the performances, this heuristic is used during high utilization (otherwise there is no relevant activity to provide the analysis performance data).

The properties of the self optimization heuristics are:

- *PerfIndicators (Performance)* – the set of performance indicators that the heuristic tries to improve. It contains a single indicator or a combined set (e.g. average query response times, calculation and aggregation times, quality of service etc.).
- *ConfigIndicators (Configuration)* - the set of configuration parameters that are changed by the heuristic with the purpose of obtaining better performance indicators (e.g. cache allocations, compression types, storage modes etc.).
- *Delta (double)* – represents the change in the configuration indicators with each step. There are two cases:
    - *Restrained/Absolute* –a possible set of values that the indicator can take, or a fixed numerical constant with which its value changes. In this case, we can speak of granularity in the situation where only certain values of the set are to be tested. (e.g. the compression mode can be either compressed or decompressed).
    - *Relative* – represents a numerical relative amount indicating also granularity of the heuristics. The smaller the delta, the higher the granularity (e.g. the indicator changes with 5% of its last value).
- *Threshold (Double)* **–** represents the threshold value for the accepted performance levels. If performances cross this threshold, the configuration changes are no longer accepted. There are two ways of defining a threshold:
    - *Absolute* **–**given by an absolute numerical value of the performance measure (e.g. 2 seconds).
    - *Relative* **–**specified as a percentage that shouldn't be crossed by the rapport between the current and previous performance levels (e.g. 5 %).

The diagram in Figure 65 presents the functioning scenario of the self optimization heuristics, for one step, for a single managed element.

- The ACM using the Self Optimization rules performs the changes on the chosen configuration indicators for the given managed element
- The Managed Element, with the new configuration, is exposed to operations from the users. At the end of the usage operations, the new performance levels are obtained.
- The ACM accepts or rejects the configuration change based on the obtained new performances.

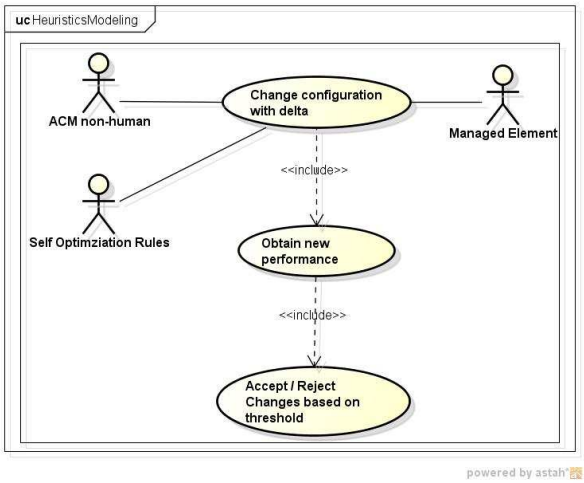


Figure 65 : General Self Optimization Heuristics for one step for one managed element

The usage of this heuristic has two roles with the data warehouses: (i) initial deployment and (ii) continuous optimization..

Initial Data Warehouse Deployment –refers the first step when putting into place a new non configured DSS. Good starting configurations are vital for DW architectures. During this period of initial deployment, the system is under continuous usage. Therefore, the notion of step is no longer related to the days, but to a set of activity tests. An example of such usage is presented below:

Example. A company decides to buy a new physical server to manage their data warehouses. The following specifications are given for the new system:

Table 22 : Initial System Deployment example

Type	**Indicator**	**Value**
Configuration given	RAM	16GB
	Hard Disk	10TB
	Logical Server	Essbase v11
	Number of bases	35
Specified Performance	Maximum avg QRT/ base	2s
	Maximum aggregation time /base	20m
Configuration required	Cache configuration / base	???
	Storage mode / base	???
	Compression / base	???
	Block size / base	???

The company needs the required configuration elements to be complemented, based on the performance specifications and the physical server's resources. Traditionally, the DSS expert who is in charge of this operation performs a set of manual operations and decides upon a configuration. With the help of our system, all he has to do is configure the starting point and wait for the stress tests to finish. The following steps are executed:

- First, the DSS expert needs activity on the system. He chooses a series of operations (reporting, calculation, restructuration etc.) that simulate the activity.
- Second, the expert defines the notion of step related to the set of queries. He decides which sets of queries are used and how many times they run. Once a run is finished, it is considered a step.
- Third, he decides how many number of steps will be performed, based on how long the step takes and how much time he has. For example, for a 2h step, he can choose to do five days of testing, meaning 60 steps, thus 60 different configurations.
- Last, he configures the delta and thresholds parameters and launches BI Self-X with the self-optimization heuristic.
- At the end of the 60 steps, the heuristic will return the best found configuration / resource allocation ratio. We remind that the value is the best found during the 60 steps and not necessarily the optimal one. Also, the heuristic doesn't guarantee that performance improves with each step. Certain changes may have disastrous results on the performances, and configuration rollbacks are often met (e.g. modifying the compression mode).

Continuous Performance Optimization –refers to modifying the configurations of an existing DW architecture. The purpose is the improvement of the performance / resource allocation ratio. Unlike the first case, performances derivation constraints are more powerful, as the system used in production, so there are users that rely on it. The set of tests in this case is the real user activity, and the notion of step is fixed to an operational day. We exemplify below, in analogy with the previous case

Example. A company wants to optimize the resource allocations for one of their physical servers. The following characteristics are given:

Table 23 : Continuous Performance Optimization example

Type	Indicator	Value
Configuration given	RAM	16GB
	Hard Disk	10TB
	Logical Server	Essbase v11
	Number of bases	35
Specified Performance	Maximum avg QRT/ base	2s
	Maximum aggregation time / base	20m
Actual Performance	avg QRT/ base	1.4s
	aggregation time / base	12m
Configuration required	Cache configuration / base	15.5GB
	Storage mode / base	ASO
	Compression / base	Compressed
	Block size / base	16KB

Based on the actual performance levels we can see that there is room for improvement, as the logical Essbase server takes all the RAM memory of the physical server. The expert configures the initial starting point and then launches the BI Self-X module, with the considerations:

- The set of queries is the real activity of the users, so he does nothing to alter this

- The step is defined as a complete operational day. During the day the bases cannot be stopped and rendered unavailable, nor can their parameters can change.
- The numbers of days to run the heuristics can be indefinite, as it doesn't require a certain period (as in the first case). The heuristic can run all the time, or until the company is satisfied.
- The expert configures the delta and the threshold in a manner that when testing a new configuration no 'harsh' performance deviations occur. Unlike the previous case, the deltas and the thresholds are always expressed as relative values.
- The expert launches the heuristics implementation program and is free to leave. Additionally, he accesses periodical reports with the evolution of the resource allocations and the corresponding performances. In this case, at the end of a certain period of time the system will find a configuration with less RAM that will fit into the specified performance objectives (e.g. 12 GB of RAM after a period of 23 days with an avg QRT / base of 1.9s).

7.2.3.4.2 Self configuration heuristic

The self-configuration heuristics are implemented by the self-configuration ACMs. Our example of self configuration heuristic is based on the following principle: *take from the rich (high performance) and give to the poor (low performance)*. This translates as the reallocation of physical and logical resources between the managed elements, according to various performance criteria. A higher level of granularity is achieved by choosing several performance categories (e.g. excellent, good, average, bad etc.).

Example. A company has two physical servers (S1 and S2) that host their data warehouses. Each server has a hard disk capacity of 10TB (5 x 2TB disks each). The load on the servers is different. The total physical size of the bases on S1 barely reaches 2TB and, with an analysis of the actual context, a DSS expert concludes there is no scenario on which sizes will be grater than 4TB. So, there is a difference of 6TB that are unused, no matter what. On the contrary, on S2 the total base sizes are 9.5TB and the analysis shows that in the nearby future there will be a need of at least 15TB. Moreover, if the space requirement is not assured, performances will drop drastically. Therefore, S1 is rich and S2 is poor. In this case, the self configuration heuristic proposes the physical action of taking 3 of the hard disks of S1 and put them on S2, if possible. This way, a disk resource reallocation has been made without any additional cost.

The self configuration heuristic formalizes the process of smart resource allocation. Instead of acquiring new (unnecessary and costly) resources, first you analyze what you can do with the actual system. We have mentioned a division between high performance and low performance managed elements. This is exclusively based on the DSS modeling of the Performance Indicators. The choice of performance indicators depends of the particular resource that is to be reconfigured (e.g. RAM cache allocations influence query response times, disk – data export/load times etc.). The properties that describe the self configuration heuristic are:

- *Physical Resources (PhysicalResource)* – the physical resources that can be reallocated by the heuristic

- *Logical Resources (LogicalResource)* – a self configuration operation can also be performed for a logical resource, even if the case is seldom met (e.g. an base is reallocated to a different logical server because the logical server imposes some constraints, like compression types, that the base cannot respect, for a good performance).
- *Performance Indicators (Performance)* – the set of managed element performance indicators that the heuristic uses when deciding if the managed element has a low or a high performance.
- *Configuration Indicators (Configuration)* – the set of managed element configuration indicators that the heuristic uses with the reallocation of resources

Figure 66 illustrates the use case for the self configuration heuristic. We remind that the heuristic is integrated by self configuration ACMs and sets of self configuration rules, with the direct impact on the managed element.

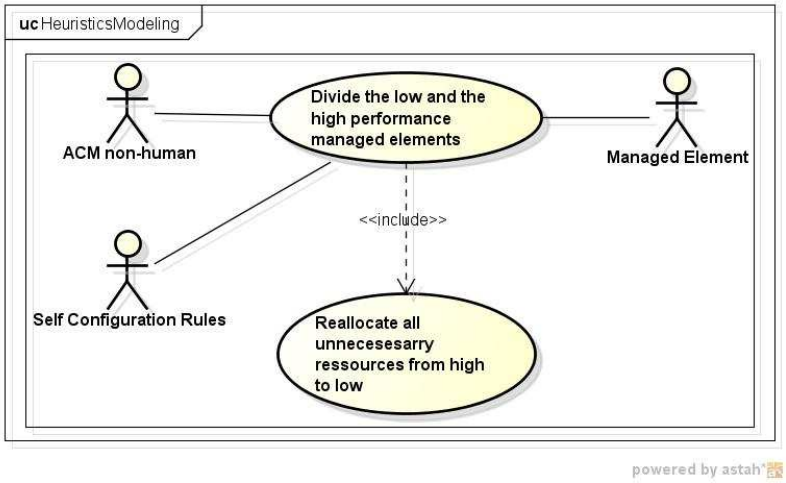


Figure 66 : Self Configuration Heuristics general diagram

- The first step is to divide the low from the high performance managed elements. Similar to the expression of thresholds from the self optimization heuristic model, there are two ways of expressing the two:
    - *Absolute* – a value representing the border of the performance limit. This is usually used when the performance requirements are explicitly described (e.g. the average QRT for a base must not be over 2s so, all bases which have an average QRT below 2s are high performance, the rest are low performance).
    - *Relative* - the threshold is computed based on the current performance levels, when no specific targeted value is given. This implies an aggregation of the performances over the DSS managed element hierarchy. (e.g. if the average QRT for a base is below the average response time of the application containing the base, then the base has a high performance; otherwise it has a low performance).

- Once the division is done, the available unnecessary resources will be reallocated among the needing managed elements. Similarly, there are two ways of making the allocation:
    - *Absolute* - the reallocation is done in fixed chunks of resources between all the low performance managed elements (e.g. 2GB of RAM memory for 4 non low performance bases results in an allocation of 500MB for each base)
    - *Relative* – the reallocation is scaled with the managed element's configuration indicators. (e.g. 2GB of RAM memory for 2 low performance bases, one that has a file size of 5GB an the other of 20GB results in an allocation of 500MB for the first and 1.5GB for the second).

On Demand Resource Reallocation - The usage purposes of this heuristic are different from the self optimization case. By analogy, the notion of step no longer has a meaning, as the self configuration is an 'on-demand' punctual operation. It can be executed at any point in time by launching the self configuration ACM on the targeted managed element (including the entire DSS).

Example. due to the drops in performance on a physical server hosting the data warehouses, a DSS expert is called to change the allocation of resources based on the actual configuration. The table below presents the current situation:

Table 24 : Self Configuration On Demand example

Type	Indicator	Value
Configuration given	RAM	16GB
	Available RAM	2GB
	Hard Disk	10TB
	Logical Server	Essbase v11
	Number of bases	35
	Low performance bases	5
Specified Performance	Maximum avg QRT/ base	2s
	Maximum aggregation time / base	20m
Actual Performance of the poor bases	avg QRT/ poor base	2.4s
	aggregation time / poor base	32m
Configuration required	Additional cache / poor base	???

The DSS expert launches an on demand request with the self configuration ACM in order to reallocate the available memory. Supposing that there is an absolute value given for reallocation, each of the poor bases will receive an additional 400MB, to allocate with their caches.

Even though they are completely independent the two heuristics are used together for better results. The self optimization heuristic provides free resources while keeping the performances at the required levels; the self-configuration heuristic makes use of these freed resources to reallocate them to the low performance elements and further improve the performances.

7.3 Transition model – from UML to DB/OWL

Prior to the presentation of the ontology adoption models, we present the consideration for the passage from UML to the ontology model (OWL and rules). We equally argue why in over the choice of ontologies for the BIOptim implementation. At this point we have a good knowledge of the problematic and of the abstract UML model that must be implemented.

The UML-ontology passage is inspired by the thesis of [Raimbault (2008)], which covers the transition between three knowledge models: UML, ontologies and conceptual graphs. The guidelines are clearly fixed and concern: class transformation, datatype property transformation, object property relation transformation, class instantiation and individuals. We have taken into considerations these transition models, and exemplify in the following how our transition is made.

7.3.1 Relational DB model – BI Copilot

The DB adoption model is the base of the BI Copilot software product. All the data of BI Copilot is represented through relational data bases. These data bases are in turn part of the input data for the ontology model. More specific, all the elements presented in the DSS static knowledge UML modeling have an equivalent in the BI Copilot data bases. The BICopilot data bases integrate both raw monitored data and aggregated data into the aggregated CMDB.

7.3.1.1 Raw monitored data

The raw monitored data comes from logs retrieved from the system. It contains the static information concerning machines resources, installed software (DSS and non DSS), data warehouses and their parameters. All these detailed elements are retrieved from different logs and are organized into data bases. We take each of the three modeling parts (architecture, configuration and performance) and present a BI Copilot simplified data base equivalence.

The DSS architecture is represented by several tables, each table corresponding to one managed element type (Base, Application etc.). The DB schema is shown in Figure 67.

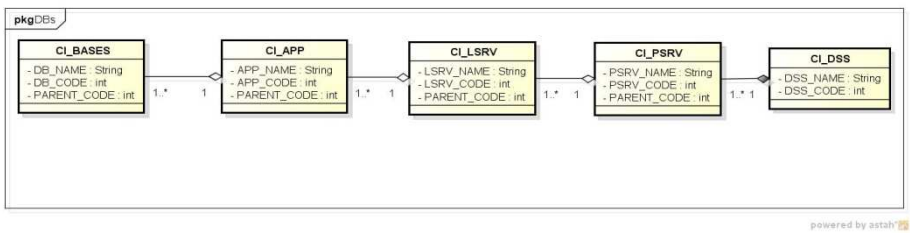


Figure 67 : BI Copilot DSS architecture hierarchy

The diagram has been simplified (in the number of columns presented) in order to prove the hierarchical architecture. Each of the management elements is linked to its direct

superior hierarchy element via the PARENT_CODE column. For example, recovering all the bases corresponding to a specific physical server implies several joins proportional to the number of hierarchical levels between the two (4 in this case).

The configuration parameters for each of the managed elements are stored in tables, with their values corresponding to the dates of their last modification and the dates at which the monitoring has been made (Figure 68)

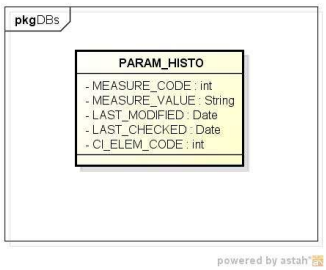


Figure 68 : DB Adoption Parameters table

PARAM_HISTO links with the CI tables, using the CI_ELEM_CODE field, which indicates to which managed element the measure applies. The MEASURE_CODE is taken from an external table of declared measures and its value is a string due to the multiple data types a measure value can have. There are two dates: LAST_CHECKED the date at which the last monitoring has been made, and LAST_MODIFIED the date at which the measure value has last changed.

The performance indicators are stored similarly to the configuration parameters, but, with taking into consideration the fact that their values change (or are prone to changes) regularly. For example, if we consider the base QRT for each performed query, by base, there are tens or hundreds of values for each day. The performance indicators are stored in the PERF_HISTO table (Figure 69).

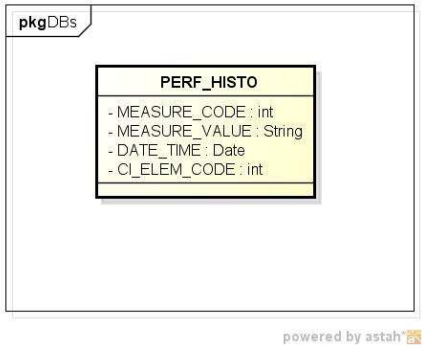


Figure 69 : DB Adoption Performance table

We notice the presence of the measure code, value and the element code that links to the CI tables, similar to the configuration table. But, this time, the two date fields are replaced with a single DATE_TIME column, which indicates the complete time at which the value for the respective measure has been recorded.

7.3.1.2 Aggregated monitored data

The second type of data stored by the BI Copilot DBs refers to the aggregated information built with the raw data. The aggregation operations are made over the configuration and performance indicators, with two aggregation axis: time and DSS hierarchy (from the CI tables). The operations include computing averages, sums or distinct values. For example, for a query response time we have the average value, for a disk occupation the sum and for the compression type the distinct values.

Time aggregation computes the specified aggregation operation over time intervals. Consider the following example.

Example. During the day of 02.06.2010, a total number of 1000 queries have been performed on a base, in the time interval from 9am to 22pm. We consider the aggregated query response time as the sum of individual query response times divided by the number of queries:

$$AvgQRT_{base} = \frac{\sum_{query} QRT_{q/base}}{no.queries}$$

Hierarchy aggregation computes the aggregation over the DSS hierarchy levels. This means that the operation is performed for each level. For operations like the sum, this doesn't affect the order of operations. But, for operations such as the average, the result is different depending on how the averages are calculated. Retaking the previous example, this time we compute the averages over the DSS architecture levels.

Example. We want to compute the average QRT for the application shown in Table 25.

Table 25 : Average computation difference for an application

Application	Base	Individual QRT (s)	Average per APP_1 (time)	Average per APP_1 (hierarchy)
APP_1	B1	1	(1+2+2+3+3)/5=2.2	[(1+2)/2 + (2+3+3)/3] / 2 = 2
		2		
	B2	2		
		3		
		3		

There is a difference of 0.2 s between the two averages. The hierarchy formula is recurrent from the lowest to the highest levels of the DSS architecture. Below we can see its result in comparison with the time formula, where Nb is the number of bases and Nq is total number of queries.

$$AvgQRT_{app/hierarchy} = \frac{\sum_{base} AvgQRT_{base}}{Nb} \leq AvgQRT_{app/time} = \frac{\sum_{base}\sum_{query} QRT_{query,base}}{Nq}$$

Note: BI Copilot ACMDBs use the time formula. In the above example, the value used is 2.2 seconds and not 2 seconds. This is a "touchy" problematic with DSS, but to give a

simple argument, using the time formula has more sense because it is directly related to the user activity.

The two ACMDBs for the PARAM_HISTO and PERF_HISTO tables are shown in Figure 70:

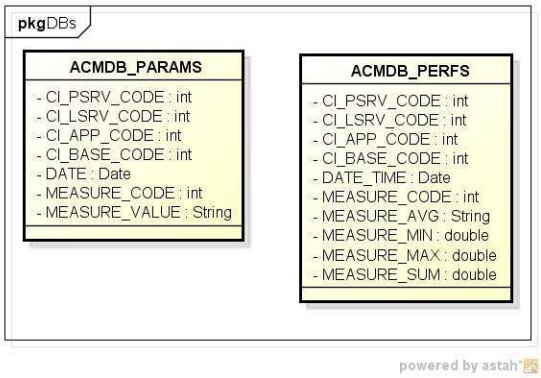


Figure 70 : BI Copilot ACMDB tables

Both tables contain the explicit DSS hierarchy and the time aggregated values of the lowest levels (here the Bases). The higher levels operations are performed starting from these tables using customized DB queries. This allows both hierarchy and time aggregation operations. The purpose of the two tables is the first level of aggregation from the raw monitored data. Any other required information can be obtained starting from this.

7.3.2 OWL ontology model – BI Self-X

The ontology adoption is a more 'delicate' subject, being in the same time an important part of the proposed approach with this thesis. The most important aspect is that the ontology model offers a complete view over the static AND the dynamic knowledge. Ontologies allow us to express the autonomic architecture, the managed element ACMs, their integration with the DSS managed elements, and, most important, the dynamics of the autonomic system, with the AC principles, MAPE-K loops and the corresponding rules.

Following the transaction considerations from [Raimbault (2008)], we identify two aspects: the passage from diagram concept to ontology classes and individuals (the taxonomy adoption), and, the passage of the links between these concepts to ontology properties (the ontology adoption).

7.3.2.1 Taxonomy adoption

The taxonomy adoption refers to the passage from the UML concepts to ontology concepts in terms of classes and instances of these classes (individuals). This is straight forward, by following the UML class and component diagrams.

Table 26 illustrates the correspondences between the two, with an example taken from the autonomic adoption model

Table 26 : OWL Ontology topology adoption correspondences

AC concept	UML	OWL	Taxonomy
MAPEStates	Class,	Owl:Class	
Diagnostics	Subclass of the states	Owl:Class	rdfs:subClassOf MAPEStates
Problems	Subclass of Diagnostics, enumeration of the problems	Owl:Class	rdfs:subClassOf Diagnostics
Errors	Subclass of Diagnostics, enumeration of the errors	Owl:Class	rdfs:subClassOf Diagnostics
NonOptimalBlockSize	Variable from Problems	Owl:NamedIndividual	rdf:type Problems
InsufficientMemory	Variable from Errors	Owl:NamedIndividual	rdf:type Errors

The correspondences are explicit:

- An UML class becomes an OWL class
- An UML subclass becomes an OWL subclass.
- The UML class variables become instances of the OWL classes (individuals).

An issue to be discussed is the instantiation of the classes, how new individuals are created especially with the dynamic aspect. The table above presented a ‘static’ part, as the instances of the diagnostics will never change (they are predefined from the enumeration classes). On the other hand, if we take the DSS hierarchy classes, each of the managed element classes will contain instances that will represent the various managed elements. These instances are created dynamically depending on the DSS architecture. This observation is also valid for the ACMs, which are directly related to the instances of the managed elements. Therefore, the adoption is done starting from the BI Copilot CI bases, with the equivalences from Table 27.

Table 27 : DB OWL topology equivalence

CI Table / line	DB concept	OWL concept
CI_APP	DB Table	Owl:Class
APP_BUDGET	DB Table Line	Owl:NamedIndividual

7.3.2.2 Ontology inter-concept links adoption

The full ontology adoption comes once the taxonomy is complete, and adds the totality of links between the classes and individuals from the taxonomy adoption. This is not as straightforward as the taxonomy adoption, because not all the correspondences are explicit. It is a more complex work which requires the intervention of an ontology architect. The main reasons behind this are the semantics of properties and the existence of deductions and inference rules.

The two types of OWL properties are used for the ontology adoption: the data type properties and the object type properties. For the *data type property* adoption, Table 28 exemplifies the correspondences. No special operations are required here. The only aspect to be taken into consideration is weather the data type property is functional (single value possible) or not.

Table 28 : Ontology data type property adoption

Instance	UML	OWL	Type
APP_BUDGET	TotalSize	owl:DatatypeProperty (xsd:Double)	Functional
APP_BUDGET	CompressionMode	owl:DatatypeProperty (xsd:String)	Non Functional

For the *object type properties*, there are several aspects to be reviewed with the ontology passage:

- *Links identification* – first step is to identify the attributes of the UML classes that link entities between them (e.g. an instance of the Physical Application class contains a list of instances from the Base class, which describe the hierarchical inclusion between the two).
- *Property functionality* **–** similar to the data type property, it must be decided whether the property has a single or multiple values.
- *Property inverse* **–** depending on the level of expression, the declaration of inverse properties is put into question (e.g. the property base isChildOf application is the inverse of application hasChild base)
- *Property symmetry* **–** which describes a property as being symmetric (e.g. the relation between an ACM and the corresponding managed element, isLinkedTo, expresses that an specific ACM individual is related to the specific Managed Element individual, and vice versa). A symmetric property is inverse to itself.
- *Property transitive* **–** expressing the formalism of transitive properties. (e.g. the hasChild property is transitive, such that the hierarchical DSS levels can link any level with any level; an instance of the Physical Server class is linked to several instances from the Base class).
- *Rule integration* **–**the true power of the ontologies, rules permit the deduction and inference of facts. The UML models present the class and component diagrams, with the associated use case scenarios. The integration of rules allows the implementation of these use cases. (e.g deciding that a server has an *InsufficientMemory* diagnostic and linking the server and its Self Healing ACM to this diagnostic)
- *Validity of the model* –the entire ontology adoption must be valid and consistent (e.g. DL).

The considerations above argue that the ontology model is richer in information than the starting UML model (e.g. Table 29).

Table 29 : Ontology object type property adoption

Instance	UML	OWL	Characteristics
APP_BUDGET	Aggregates	Owl:ObjectProperty (instance of Base)	Transitive
APP_BUDGET	Direct link	Owl:ObjectProperty (instanc of ACM)	Functional, Symmetric
APP_BUDGET	Aggregates	Owl:ObjectProperty (instance of Diagnostics)	

Note. With the adoption of ontology models, one difficult part is the implementation of mathematical operations. For example, calculating the average query response time for the physical server by using only the elements that are in the ontology proves to be a difficult task. Therefore, we use two alternatives to overcome this.

- *DB passage* – if the BI Copilot ACMDBs are available, we connect to them, compute the desired value with DB queries (AVG, SUM etc.), return the result and update the ontology properties (Figure 71).

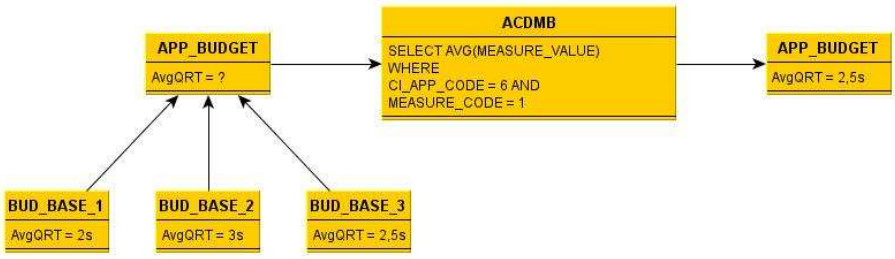


Figure 71 : Computing ontology averages with the ACMDBs

- *ARQ queries* – the BI Copilot ACMDBs are not available; in this case we make use of update queries, ARQ SPARQL Update, which compute the aggregation operation over the hasChild DSS hierarchy property.

Figure 72 : Computing ontology averages with ARQ queries

Nevertheless, in both cases, the ontology model must be revalidated after the modification operations occur.

7.3.2.3 Rule construction

The last stage of the transformation is the construction of the ontology rules. This is the most delicate aspect of the passage. With UML we have presented the individual rule

representation and the system rule organization. The question that remains is how is the content of the rule (i.e. its statement) transformed into usable knowledge?

The first part of rule integration, OWL transformation and system rule organization, is done in a similar way with the previous elements (Table 30):

Table 30 : Ontology rule representation

UML	OWL	Ontology concept
Class Rule	rdf:type	Instance of Rule
Rule phase	rdf:type	Instance of the four subclasses of Rule (e.g. RuleAnalyze)
Rule purpose	Owl:ObjectProperty	Instance of Purpose
Rule Name	Owl:DatatypeProperty	xsd:string
Rule Body (can be a link to a file)	Owl:DatatypeProperty	xsd:string
Corresponding ACM	Owl:DatatypeProperty	Instance of ACM

The second part, the actual rule integration with the system is discussed in the next section as part of the semantics implementation. The UML representation of rules doesn't integrate the understanding of the rule or its impact, only its formal representation. A rule cannot be built in an automated way. It requires a prior analysis to what it states and the intervention of a human expert who understands both the rule content and the ontology model.

7.4 Semantics modeling and implementation

Following the previous two sections on UML modeling and the UML-Ontology transition, this section presents how the proposed models are implemented using ontologies and ontology based rules.

We have seen from the state of the art that ontologies are a viable technology for modeling and expressing knowledge. We first present an overview of the used technologies for this model. Then, we retake the UML odel, already having the modeled elements introduced, and focus purely on the ontology model. We give arguments, where the case, why this technology is suited for the approach; in the same time we also point out its drawbacks, as to have a clear and objective view of the consequences from adopting it.

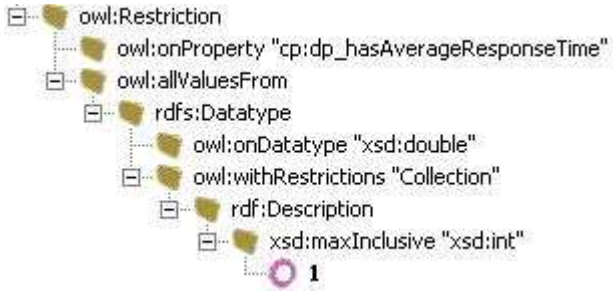
7.4.1 Chosen technologies and tools

The state of the art presented the main semantic technologies and the tools for working with them. From this "pool" we have chosen the following elements for our approach.

OWL DL: used for knowledge description (DSS model and AC adoption model). The choice came from:

- The need for mathematical expressivity with the data type properties; we need to express a lot of mathematical constraints and operations.

Example. The following constraint needs to be represented: if a managed element is a base and its average query response time is above 1s, then it should be classified as a non performing base. With OWL 2 this kind of data type restriction is possible (illustrated with the help of XML Viewer [MindFusion (2010)]):



- It is the most expressive semantic language that still keeps the possibility of reasoner integration. Even if OWL Full has the full expressivity (that sometimes we require) we are 'forced' to pick the next best thing, as the DL combines the most of the expressivity and reasoning support.
  Example. We want to express the fact that an instance of the Physical Server contains all the individuals from the Base class. With OWL Full this is expressed with the triplet *<i_MyPhysicalServer, hasChild, c_Base>* and doesn't support inference. With DL we use an inference rule that allows to link the *i_MyPhysicalServer* individual with all the individuals from the *c_Base* class: *(?i_base rdf:typ c_Base) → (i_MyPhysicalServer hasChild ?i_base)*

For the modeling of the ontology based rules we have chosen Jena Rules and SPARQL ARQ Update queries.

Jena Rules: we have adopted this type of rules to model some of the dynamic knowledge base. Several arguments lead to this decision:

- Need for inference support with the rules, more precisely with the topology classification.
  Example. For a physical server, we want to propagate all the diagnostics from the managed elements that are below it in the DSS hierarchy. The asserted ontology model doesn't allow this deduction. With a Jena rule we have:

  *[**Rule 1:***
  (i_MyPhysicalServer rdf:type c_PhysicalServer)
  (?i_me rdf:type c_ManagedElement)
  (?i_me hasDiagnostic ?i_diag)
  (i_MyPhiscalServer hasChild ?i_me) →
  (?i_MyPhysicalServer hasDIagnostic ?i_diag)]

  The *rdf:type c_ManagedElement* triplet performs the match on the inferred ontology model (thus all instances: Bases, Physical Applications etc.).
- The need of rule dynamics, as seen from the dynamic knowledge base modeling. There is a requirement of executing sets of rules, at given points with interdependent results (e.g. the MAPE-K loop). Jena rules offer a high flexibility

with expressing entailments (unlike SWRL for instance, where rules are executed all at once).

Example. the expression of the following self healing diagnostic rule entailment: if an application has more memory in the base caches than the actual RAM memory, then reconfigure the cache allocations. First, we use a rule that verifies the first condition and establishes the diagnostic (*MemoryOverload*), and then a rule that based on this diagnostic proposes a reconfiguration heal.

*[**Rule 1:***
(?i_app rdf:type c_PhysicalApplication)
(i_MyPhiscalServer hasAvailableMemory ?i_psrv_mem)
(?i_app hasOccupiedMemory ?i_app_om)
greaterThan(?i_app_om, ?i_psrv_mem) →
(?i_app hasDiagnostic MemoryOverload)]

*[**Rule 2:***
(?i_app rdf:type c_PhysicalApplication)
(?i_app hasDiagnostic MemoryOverload)]
(i_MyPhiscalServer hasDiagnostic ?RAMLimitReached) →
(?i_app hasHeal ReconfigureCaches)]

- Last, Jena rules assure the consistency of the ontology model and don't infer on inconsistent models.

  Example. For a base, the storage mode can be either ASO or BSO. In the ontology model, the ***hasStorageMode*** property can have only two values "ASO" and "BSO". Anything other value renders the property domain inconsistent. A rule that which modify this property is forced by a validity check to ensure that the model remains consistent.

ARQ SPARQL Update: comes as a replacement to certain Jena rules, in the situations where:

- Mathematical operations are expressed. With Jena rules, the expression of mathematical formulas is limited, as well as complex operations such as aggregation or count.

  Example. We need to express the following rule: *a physical server redistributes equally the available RAM memory to the existing applications on the server.* With Jena rules this is impossible as there is no notion of the number of applications that a physical server has (a count on the *hasChild* property). With SPARQL Update we can express the count along with the group by clause. The query below computes the number of applications for each physical server based on the *hasChild* property.

```
SELECT COUNT(?i_app) ?i_psrv
WHERE
{
    ?i_psrv hasChild ?i_app.
    ?i_app rdf:type c_PhysicalApplication.
    ?i_psrv rdf:type c_PhysicalServer
}
GROUP BY ?i_psrv.
```

- Alternative conditions are expressed. The usage of IF equivalent expressions with Jena rules ontology rules forces the duplication of big rule portions, as the Jena rule engines doesn't take into account the OR logical operator. The SPARQL FILTER operator provides support for this drawback.
  Example. We want to answer the following question: *which bases have the average query response time lower than 2s OR greater than 4s*. The corresponding query with SPARQL is:

```
SELECT ?i_base
WHERE
{
      ?i_base rdf:type c_Base.
      ?i_base hasAverageQueryResponseTime ?qrt.
      FILTER (?qrt < 2.0 || ?qrt > 4.0)
}
```

Protégé 3.4 & 4 was the selected GUI tool for working with ontologies. The choice in this case was straight forward, as Protégé proved to be a viable GUI (especially in its 4th generation) as well as a community highly supported tool. Thus, the model images that are shown in the ontology modeling are extracted from it with the help of the Jambalaya plugin. Of course, as we treat xml file based ontologies, we have also made extensive use of text editors for direct ontology modification.

The overall schema of the ontology model contains four ontologies, organized by their purposes, and which import one another as shown in Figure 73.

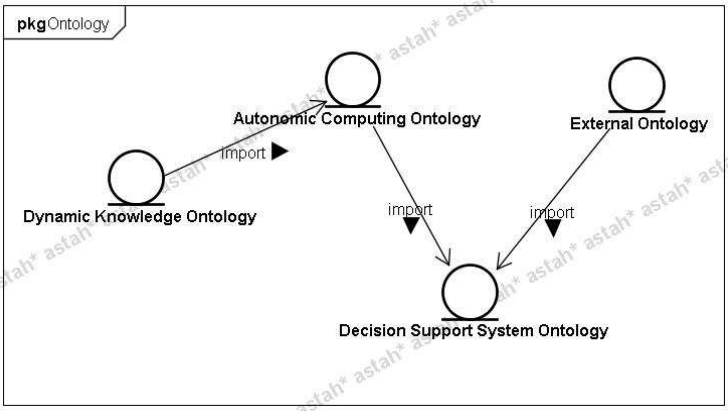


Figure 73 : The General Ontology Dependences

- *External Ontology* – corresponding to the external UML model, with information about the users of the DSS, the existing companies and software editors. It is treated in a different ontology, unlike in the UML model.
- *Dynamic Knowledge Ontology* **–** representing the dynamic knowledge, with the expression of the ontology rules.
- *Autonomic Computing Ontology* **–** the modeling of the autonomic computing model, with the principles, states, diagnostics and heals corresponding to the MAPE-K

loops. It imports the Dynamic Knowledge Ontology as it makes use of the existing rules.

- *Decision Support System Ontology* – the core ontology that expresses the DSS. It imports both the External and the Autonomic Computing Ontology (and by transitivity the Dynamic Knowledge Ontology).

We mention that in the following subsections we focus solely on the ontology modeling aspect, as the description of the modeled elements has already been done in the UML model.

Note. Regarding the naming conventions, we have used a series of prefixes to identify directly from the name the type of the ontology concept which is manipulated. These conventions are chosen by us to facilitate the work with the ontology (in the GUI interfaces, in the backend development and throughout this thesis by facilitating comprehensibility):

- *"c_"*: used for the any owl:Class concept
- *"i_"*: used for any owl:NamedIndividual concept
- *"op_":* used for any owl:ObjectProperty concept
- *"dp_":* used for any owl:DatatypeProperty concept

7.4.2 External ontology

The concepts of the external ontology are shown in Figure 74:

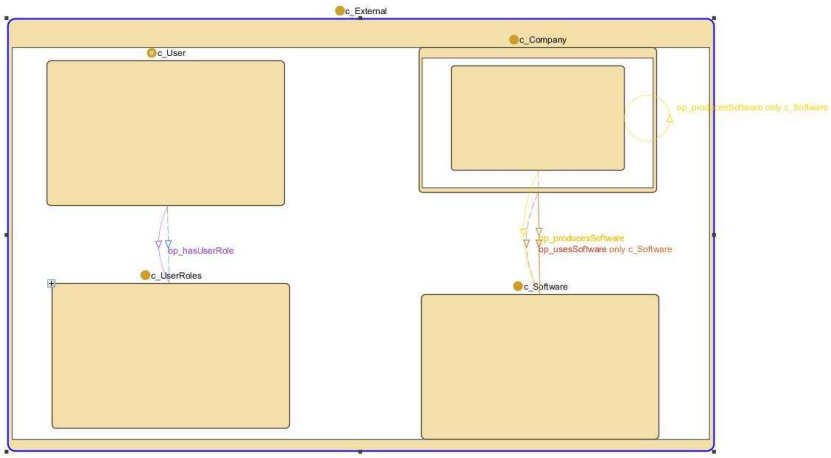


Figure 74 : External Ontology Elements

- *c_User* – defined class, with the restrictions that any of it instances should have a *c_UserRole*, a user name and a password:
  - *(op_hasUserRole only c_UserRole)* (necessary & sufficient)
  - *(dp_hasUserName only string) and (dp_hasUserPassword only String)* (necessary & sufficient)
- *c_UserRoles* – non defined class with the list of instances describing the possible user roles, e.g:
  - *(i_UserRoleExpert rdf:type c_UserRoles)*

- *(i_UserRoleResponsable rdf:type c_UserRoles)*

- *c_Company* – generalized class with the restriction that it must use at least one software product:
  - *(op_usesSoftware only c_Software)* (necessary)
  - *(op_usesSoftware min 1)* (necessary)
- *c_SoftwareEditor* –subclass of a company, with the restriction that a software editor produces at least one software
  - *(rdfs:subClassOf c_Company)* (necessary)
  - *(op_producesSoftware only c_Software)* (necessary & sufficient)
  - *(op_producesSoftware min 1)* (necessary & sufficient)
- *c_Software* – a generalized class that contains the instances of various software. For future implementations this should make direct use of existing software ontologies (one of our future work projects). Nevertheless, for the present work, it containsonly asserted instances, such as:
  - *(i_Windows2003ServerP2 rdf:type c_Software)*
  - *(i_HypEssbase11.1.1 rdf:type c_Software)*

7.4.3 DSS ontology model

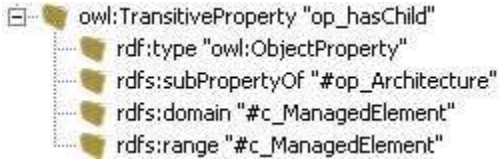
The section is split into three parts, similar to the UML DSS model. In addition, the detailed description of SLA/Os implementation is also shown.

Note. We remind that the DSS ontology model provides the description of the DSS at a precisely given time, thus lacking the temporal axis (it can be seen as a photo of the system). The place where this temporal axis is required is in the dynamic knowledge with the implementation of the self-management heuristics. The solution in that case is to use a 'previous' trace of the concerned properties (e.g. *dp_hasIndexCachePrev*).

7.4.3.1 DSS architecture

The DSS architecture classes are shown in Figure 75. The representative concepts are:

- *op_hasChild* – transitive object property describing the hierarchical links between the managed elements of the DSS (and its corresponding inverse property - *isChildOf* such that links are recovered in both ways).



- *"some" restriction* - Each managed element class has a restriction regarding its direct children. The use of "some" and not "only" is explained by the transitivity of the *op_hasChild* to ensure genericity. For instance, for the *c_PhysicalServer* class we have
  - *(op_hasChild some c_LogicalServer)* (necessary)

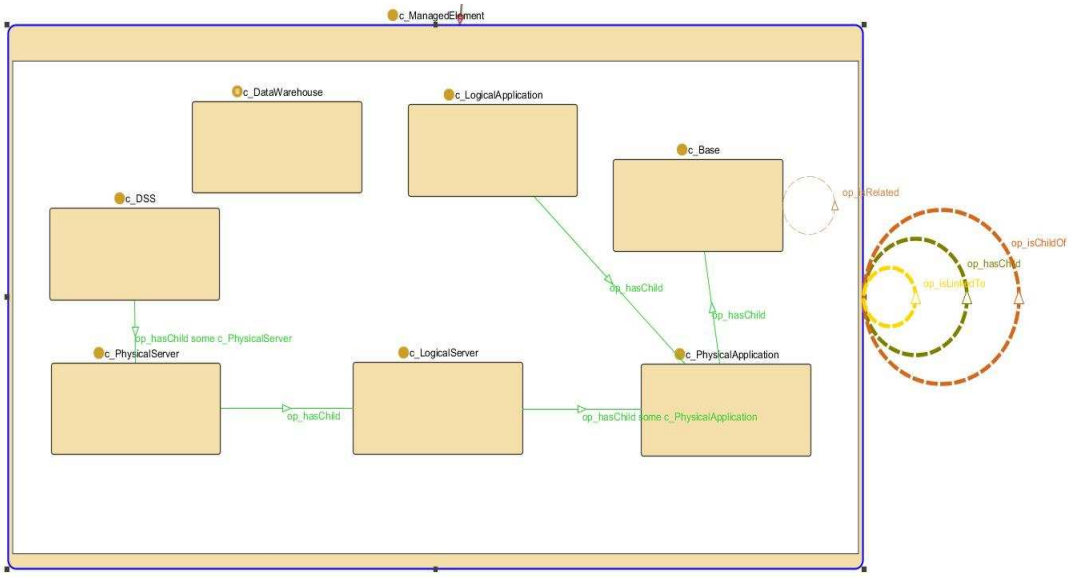


Figure 75 : The DSS Ontology class model

- *c_DataWarehouse* – defined class, its restrictions indicating the existence of at least one *c_Base or c_PhysicalApplication* instance as a child:
  - *(op_hasChild only c_Bases) or (op_hasChild only c_PhysicalApplication)* (necessary & sufficient)
  - *(op_hasChild min 1)* (necessary)
- *c_LogicalApplication* – defined class, linking to groups of physical applications. On a meta-level, a *c_LogicalApplication* can be equivalent with a *c_DataWarehouse*, although they are different concepts.
  - *(op_hasChild only c_PhysicalApplication)* (necessary)
  - *(op_hasChild min 1)* (necessary)

From the construction point of view, there are no manually asserted instances for the DSS hierarchy. They are automatically created from the CI bases of BI Copilot. Once the ontology is populated, a consistency check is performed to ensure that further inference operations are possible. An example of inconsistency due to a bad loading process is exemplified below:

Example. An undetected mistake in the BI Copilot DBs inverses the unique IDs of a base with an application. The result is that in the DBs the fault base is shown as being the parent of other existing bases. This is not detected as no validity check is made over the DBs. When loading the ontology model, the fault base is created in the *c_Base* class.When the consistency check is done there is a violation of the *c_Base* restriction (as a base cannot have for a child another base). This way, the problem can be corrected at its root, in the BI Copilot DBs.

7.4.3.2 Managed elements parameters

The ontology modeling of the managed element parameters (both configuration and performance) is done in the same manner. These properties are mainly expressed with the

usage of *owl:DatatypeProperties*. The ontology model permits the organization of the properties in a hierarchical manner, thus allowing groups of properties to be defined by their purpose. Figure 76 presents a part of the parameter ontology model.

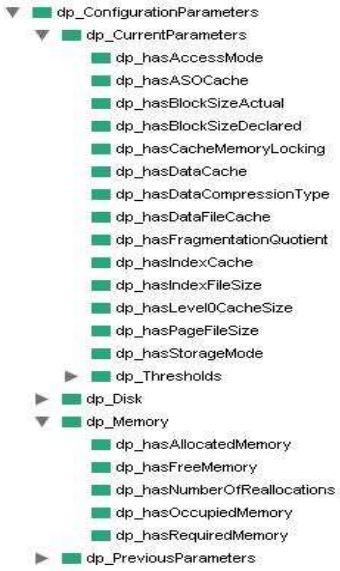
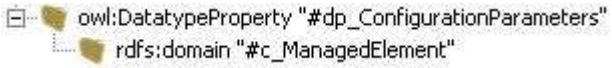


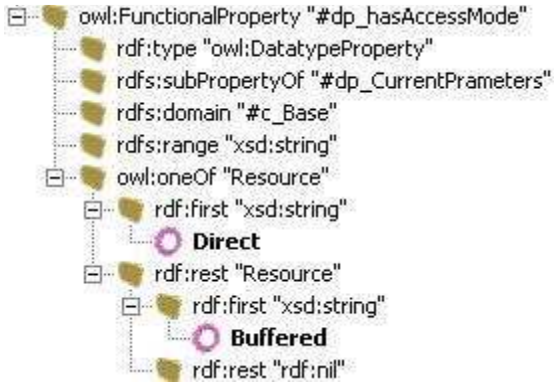
Figure 76 - Configuration properties ontology hierarchy

- *dp_ConfigurationParameters* - the root property that regroups all the values for the configuration parameters, including the information regarding system resources. Its domain is the general *c_ManagedElement* class. The domains of its sub properties are specified as sub classes of *c_ManagedElement* depending on the specific managed element which the property concerns.



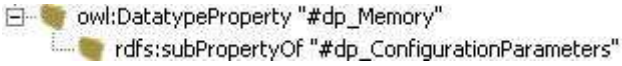
- *dp_hasIndexCache* – the data type property expressing the value of the index cache for a base. Because the property aggregates over the DSS architecture, the domain rests the *c_ManagedElement* (inherited from the root property). There can be only one index cache value at a time, therefore the property is functional.

- *dp_hasAccessMode* – similar to the previous property, with the distinction that the possible values are restricted to a limited value pool. The possible values for *dp_hasAccessMode* are either “Direct” or “Buffered”. The property is restricted to the base class only, and is not present to other *c_ManagedElement* instances.



- *dp_hasCacheMemoryLocking* – in this case of properties, which express the existence of a certain feature, the datatype property is defined over a range of boolean values.

- *dp_Memory* – the parent property for all the memory related properties. It has no asserted values, and is a sub property of the root property *dp_ConfigurationParameters*. Its domain is inferred from the root property as *c_ManagedElement*.



We have exemplified above the main data type property categories that are used. Every other managed element data type property is described in a similar manner.

Note. Due to the hierarchical representation of the data type properties, the values will always be asserted on the lowest levels (the leaves). Therefore, properties such as *dp_ConfigurationParameters* or *dp_Memory* will never have asserted values. The values that they have are always aggregated from their leaves (aggregation in the sense of adding all existing leaf values).

Example. A instance of the *c_PhysicalServer* class has two asserted property values, *(i_MyPhysicalServer, dp_hasAllocatedMemory, '4094'^^xsd:double)* and *(i_MyPhysicalServer, dp_hasOccupiedMemory, '3000'^^xsd:double).* These are filled from the BI Copilot DBs. The ontology model will aggregate these to the *dp_Memory* property thus two additional triplets will be added by the ontology: *(i_MyPhysicalServer, dp_Memory, '4094'^^xsd:double)* and *(i_MyPhysicalServer, dp_Memory, '3000'^^xsd:double).*

The performance indicator ontology modeling is similar to the configuration model. The performance indicators are part of the same data type property categories, with no exception.

7.4.3.3 Service levels and objectives agreements – SLA/O

One important managed element aspect that has a 'special' treatment from the ontology point of view is the *SLA/Os*. These are asserted directly in the ontology based on the client specifications (thus they are not filled from the BI Copilot DBs). Moreover, the BI Copilot DBs contain no information whatsoever regarding SLO/As. This information comes from external sources (i.e. the documents of the license agreements) and is formalized in the ontology only. To this end, SLA/Os are treated apart in a sub ontology of the DSS model. Figure 77 shows the SLA ontology.

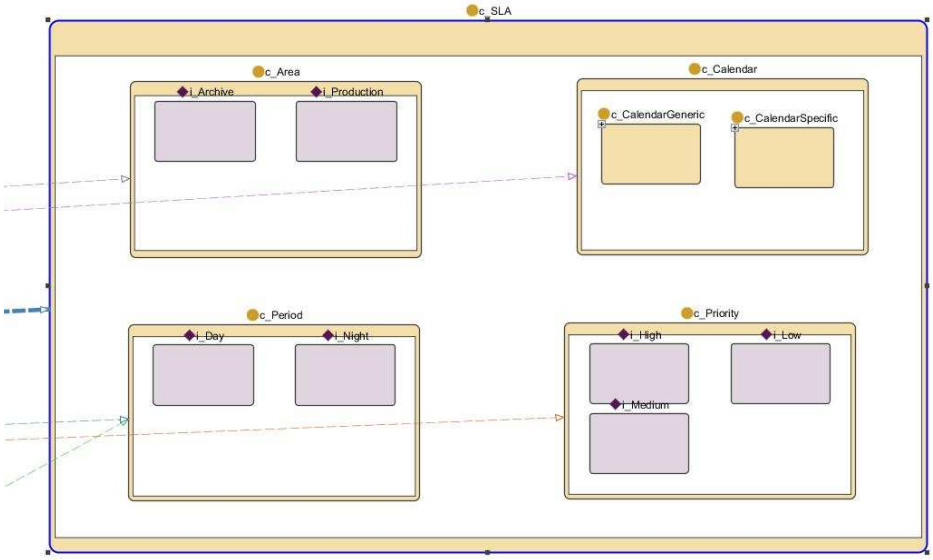


Figure 77 : SLA ontology

- *c_Period* – an undefined class, which contains two instances describing the usage period (day/night) associated with the managed elements. Similar, the *c_Area and c_Priority* classes define the types and the priorities a managed element has.
- *c_Calendar* – the utilization period ontology, instantiating all the possible utilization periods. A calendar instance can be classified as either generic or specific. The *c_Clendar* class is defined with the following restrictions:
    - *(dp_hasCalendarDay min 1)* (necessary & sufficient)
    - *(dp_hasCalendarInterval exactly 1)* (necessary & sufficient)
    - *(dp_hasCalendarIntervalValue min 1)* (necessary & sufficient)
    - *(dp_hasCalendarYear min 1)* (necessary & sufficient)
- *c_CalendarGeneric* – contains three instances that describe the current time model. Specifically, it contains the instances for: the current day and time (when the ontology is constructed), the day calendar period and the night calendar period. In addition to the four inherited restrictions form *c_Calendar*, two additional restrictions are defined for generic calendars:
    - *(dp_hasCalendarDayTimeEnd exactly 1)* (necessary & sufficient)
    - *(dp_hasCalendarDayTimeStart exactly 1)* (necessary & sufficient)

- *c_CalendarSpecific* – the second sub class of *c_Calendar*, which contains the utilization periods linked to the managed elements..

The ontology model of a calendar divides the time periods into several parts. This is mainly due to the repetition of utilization periods A calendar period specifies:

- the year(s) concerned by the period
- the interval value which can be either week (W) or month (M)
- the list of values of the interval corresponding to its value: 1-52 if week, 1-12 if month
- the list of days, depending to the interval value: 1-7 if week, 1-31 if month

To better understand the modeling of the *c_Calendar* class, we consider the following example.

Example. An application is specified as being in use *the first 5 days of each month, from January to April 2010*. The OWL representation of the corresponding calendar instance of this specific period is:

This modeling permits the definition of recursive time periods, ad also allows a complete control when working with utilization periods. A modification in a utilization period means a modification to the values of the *c_Calendar* instance and that is all. The linked managed elements will afterwards automatically modify their importance accordingly.

7.4.4 Autonomic computing adoption ontology model

The second ontology model suits the autonomic computing adoption, over the DSS model. From the UML model there are two aspects which are translated:

- The AC ontology, with the hierarchy of the ACMs, loop phase expression etc.

- The ‘K’ from the MAPE-K loop (the ontology model presented earlier)

7.4.4.1 Autonomic computing architecture

The organization of the AC architecture is similar to the DSS model, with an ontology of the ACMs. The difference is that in this case the presence of instances at lower levels is compulsory if a superior level is present. Looking back, in the DSS model, a logical server could have had a base as a child directly, without requiring the existence of a physical application. In the AC model, an instance of the *c_ManualACM* requires at least of instance of the *c_OrchastratedACM*, *c_TouchPointACM* and *c_ManagedElementACM* (Figure 78).

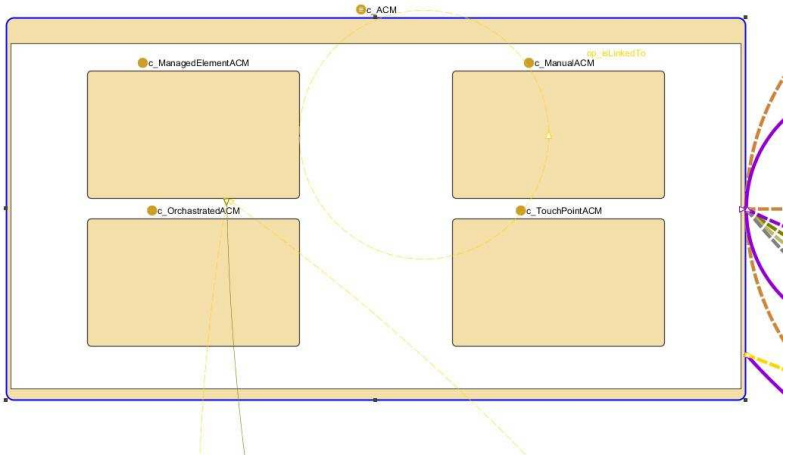
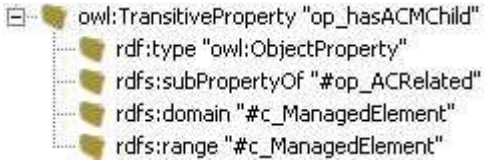


Figure 78 : The ACM ontology

- *op_hasACMChild* –the transitive object property that links one *c_ACM* with another. It is basically the equivalent of *op_hasChild* property from the *c_ManagedElement*.



- *op_isLinkedTo* – the object property that connects a *c_ManagedElement* instance with its corresponding *c_ACM* instance. The property is symmetric and non functional.

- *c_ACM* – the defined class for all the ACM instances. A restriction is given for this parent class, such that any *c_ACM* instance must have a specified purpose (from the four ACM principles):
    - *(op_hasPurpose only c_Purpose)* (necessary & sufficient)
    - *(op_hasPurpose exactly 1)* (necessary & sufficient)
- *Sub classes of c_ACM:* are defined with consideration to the AC hierarchy. For example, the *c_TouchPoint* class is defined as:
    - *(op_hasACMChild some c_TouchPointACM)* (necessary)
    - *(op_hasACMChild min 1)* (necessary)
- The exception from the restrictions above is the lower level of the AC hierarchy, the *c_ManagedElementACM*, on which the restriction is that it should always be linked to an existing managed element:
    - *(op_isLinkedTo only c_ManagedElement)* (necessary)
    - *(op_isLinkedTo min 1)* (necessary)

7.4.4.2 Managed element autonomic computing manager

In the managed element ACM ontology model the four phases of the MAPE-K loop are implemented by introduction of the corresponding rules. An instance of the *c_ManagedElementACM* presents several restrictions: be linked to at least one managed element, have an AC purpose, have a list of corresponding rules, specify whether it is active or not. Formally, these are:

- *(op_isLinkedTo only c_ManagedElement) and (op_isLinkedTo min 1)* (necessary & sufficient)
- *(op_hasPurpose only c_Purpose) and (op_hasPurpose exactly 1)* (necessary & sufficient)
- *(op_hasRules only c_Rules)* (necessary & sufficient)
- *(dp_isACMActive exactly 1)* (necessary & sufficient)

The managed elements make extensive use of the rest of the AC ontology (i.e. the purposes, the diagnostics, the heals etc), as shown in Figure 79.

We observe the loop phases (*c_LoopPhases*) and the AC principles (*c_Purpose*), as undefined classes with their asserted instances. The *c_LoopPhases* are directly related to the *c_Rules* (which will be detailed in the dynamic KB), reinforcing the fact that the phases of the managed element ACM are modeled indirectly by grouping the rules belonging to each loop phase. The *c_Purpose* class contains the four principles to which any instance of the *c_ACM* relates to. The *c_EffectorActions* instantiates a list of possible actions that can be taken as part of the effectors, at the end of the MAPE-K loop.

The *c_States* class is a sub ontology on its own, as it models the managed element's loop inter-phase states. The modeling defines the concepts, their deduction being assured by the loop rules from the dynamic KB. The diagram above showed that the *op_hasState* object property links an *c_ManagedElement* instance with one or several *c_State* instances. A detailed view of the states is shown in Figure 80.

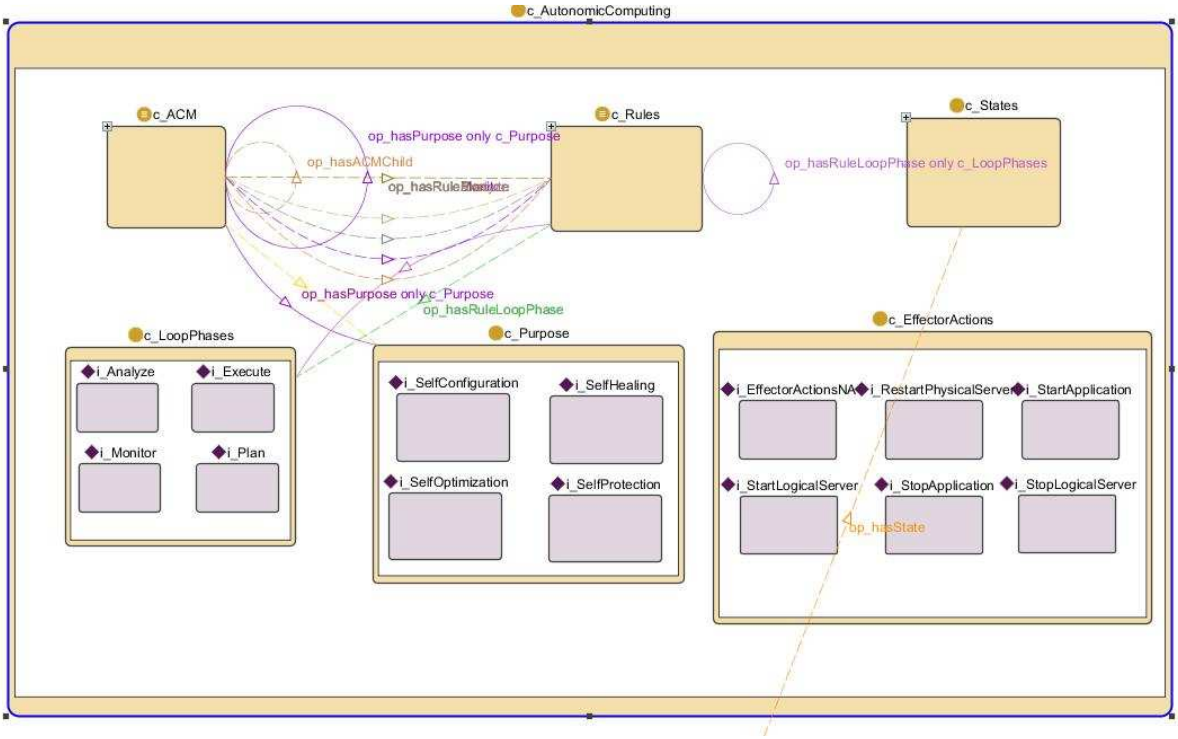


Figure 79 : AC ontology model

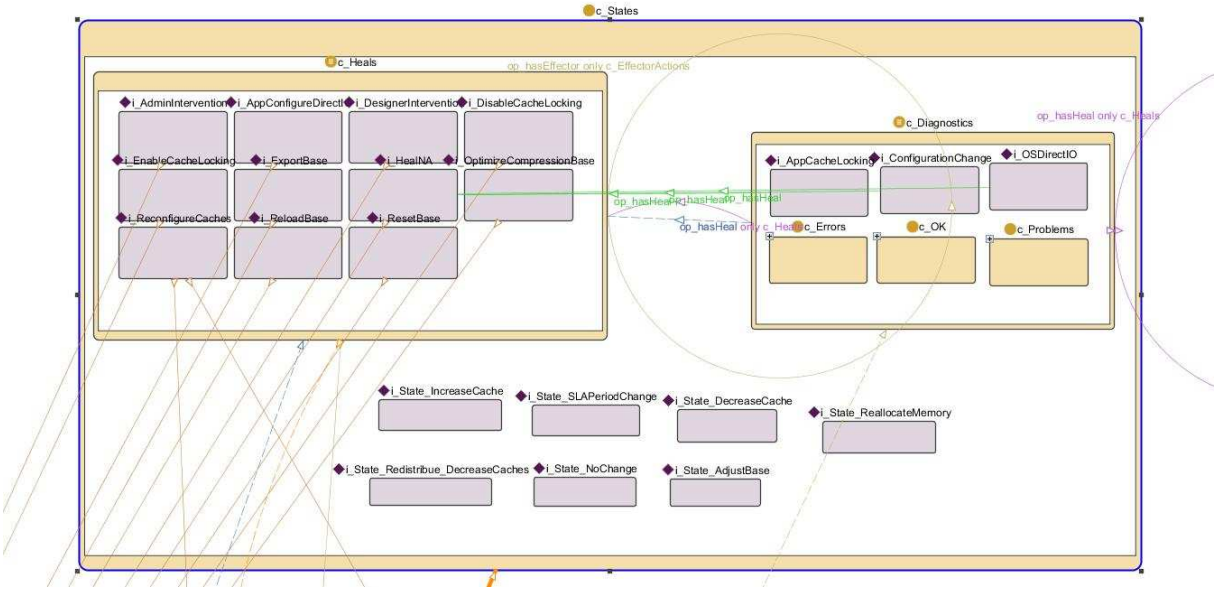


Figure 80 : The states ontology

The states are organized into:

- *generalized states* (the direct instances of *c_States*): e.g. *i_State_NoChange*
- *c_Diagnostics* – a defined class indicating various diagnostics. It contains three sub classes: errors, problems and ok. The restriction on the diagnostics is that each diagnostic instance must be linked to one or several *c_Heal* instances (even if is generic or not known: *i_HealNA*):

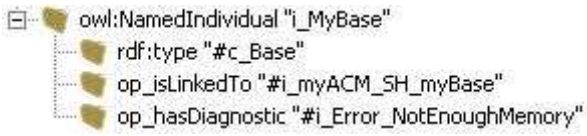
(op_hasHeal only c_Heals) and (op_hasHeal min 1) (necessary & sufficient)

- *c_Heals* – the defined class that instantiates the possible heals that a diagnostic may have. What defines a heal is that it is linked with an effectors' action and it is specified as either autonomic or not:
  (op_hasEffector only c_EffectorActions) (necessary & sufficient)
  (dp_isHealAutonomic only boolean) and (dp_isHealAutonomic exactly 1) (necessary & sufficient)

Having the complete ontology model, we understand how a managed element ACM is modeled in the ontology with the following example.

Example. For a *c_Base* instance, we describe an ACM., with all the elements presented above. It assures the self healing principle, links to a series of rules for each phase, and describes the state in which the manager is (active or not)

Following the rules for the ACM, the *i_MyBase c_Base* instance is described as:



The heals and actions are then deduced from the *i_Error_NotEnoughMemory* diagnostic. In this case, the heal is to reconfigure caches (*i_ReconfigureCaches*), is autonomous and depends on the effectors actions to stop and start the base.

7.4.5 Dynamic knowledge base ontology implementation

The dynamic KB ontology implementation is the last part of the ontology model. It details the knowledge source and rule ontology model. In fairness to the logical chaining of the section, we first detail the organization of the knowledge source, and then how the rules are modeled and how do they punctually affect the ontology. Afterwards we focus on the rule temporal aspect and exemplify over the two heuristics.

7.4.5.1 Knowledge sources

The knowledge sources are modeled as instances of the *c_KnowlegeSource* class, defined by:

- *(dp_hasSourcePath min 1)* (necessary & sufficient)
- *(dp_hasSourceType min 1)* (necessary & sufficient)

The construction of a separate knowledge source ontology is part of a parallel project developed by SP2, and is much more detailed in that case. For the BI Self-X project, the needs are assured by the existence of simple instances, which identify the source from where a piece of information comes from.

The knowledge source ontology model is simple; the only separation is done by source type, defining the three types (document, web and expert) as defined sub-classes:

- *c_KSDocument – (dp_hasSourceType only "Document"^^xsd:string)* (necessary & sufficient)
- *c_KSWeb – (dp_hasSourceType only "Web"^^xsd:string)* (necessary & sufficient)
- *c_KSExpert – (dp_hasSourceType only "Expert"^^xsd:string)* (necessary & sufficient)

7.4.5.2 Rules

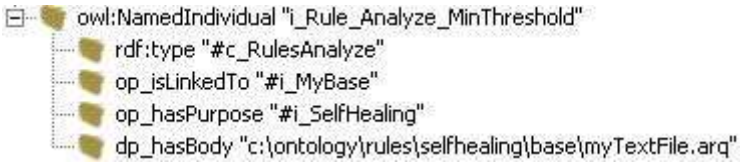
Rules are expressed in the ontology as instances of the *c_Rules* class from the AC model. The class has four defined disjoint sub classes, which classify the rules by the MAPE-K loop phases. The *c_Rules* class is defined with restrictions over the properties taken from the UML model: the corresponding loop phase, the source of the rule or the ACMs that make use of this rule.

- *(op_hasRuleLoopPhase only c_LoopPhases)* (necessary & sufficient)
- *(op_hasRuleSource only c_KnowledgeSource)* (necessary & sufficient).

Concerning the four subclasses, for instance, the class that classifies the analyze rules, *c_RulesAnalyze*, has the restriction: *(op_hasRuleAnalyze op_hasRuleLoopPhase i_Analyze)* (necessary & sufficient) along with the inherited restrictions from the *c_Rules* class.

Each rule is linked to its body (or the corresponding file that describes its body). The datatype property *dp_hasRuleBody* contains a string value that points to this information (most of the cases it is a path towards the location of a rule file), as exemplified below.

Example. An instance of the *c_RulesAnalyze* class, expressing the diagnostic: *the minimum base index cache values are not respected*, rule which is in turn described in a text file:



The body of the rule is expressed with an ARQ SPARQL Update query, in the *myTextFile.arq* file. It adds the *i_Error_MinIndexCache* diagnostic for all bases which have the value of the index cache smaller than the specified minimum threshold.

```
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
        ?base cp:op_hasDiagnostic cp:i_Error_MinIndexCache
}
WHERE
{
    ?base rdf:type cp:c_Base.
    ?base cp:dp_hasStorageMode "BSO".
    ?base cp:dp_hasIndexCache ?val.
    ?base cp:dp_hasIndexCacheMin ?val_min.
    FILTER (?val < ?val_min).
}
```

We mention that a file can contain several rules, executed in the order of appearance (top-down).

This modeling allows a clear organization of a high number of rules. We have seen how rules split based on their purpose and links to the ACMs. We explain next the process of keeping up to date (adding/modifying/deleting) the rule knowledge base. It is composed of two sub-processes: (i) translation of a rule with the elements from the ontology and (ii) assessment of the new rule impact with the existing rules.

7.4.5.2.1 Rule translation

Refers to the translation of a non structured piece of information from any of the three knowledge source types to one or several structured ontology rules. Translation is maybe the most non-trivial element of the approach, as it is strongly coupled with the DSS ontology model. This is why we tried to render the DSS ontology model as generic as possible, such that it suites any DSS architecture. The AC ontology model is generic at the same time as it is not dependent on the DSS software implementation..

In our modeling, rule translation implies the existence of a human expert, with perfect knowledge of the ontology models (including the existing rules) and with the capability of understating the expression of the piece of information described in the non structured source. He doesn't require a full understanding of the DSS, but rather the corresponding notions. For instance he may not understand what an index cache is used for, but only requires understanding that it is a configuration parameter expressed in MB and corresponds to a base. The technical details of the purpose of the index cache are not required by the person who makes the translation. There are two scenarios with rule translation: an easy one and a hard one.

The easy scenario - when faced with a new knowledge source, the ontology concepts for the expressed pieces of information already exist. This means that only a translation of the rule is required with the existent concepts. The steps of the translation are, in this order:

- Get the knowledge source and identify the pieces of information that are likely to become rules of the system (human expert)

- Identify the generic correspondences for this specific rules (human expert or advanced NLP)
- Create the rule body using the existing ontology concepts
- Create the rule instance, and specify the rule properties according to the corresponding *c_Rule*s subclass:
    - Rule body
    - Rule source
    - The linked managed elements
    - Rule purpose
- Assuring that any eventual ontology changes do not render the model inconsistent by doing a validity check

Example. One post in the technical forum at (http://forums.oracle.com/forums/forum.jspa?forumID=405) states that: for *Hyperion Essbase bases, with sizes grater than 1GB, the minimum index cache values is no longer 3MB but 10MB*. The correspondences of the specific rule – generic ontology concepts are:

- Essbase Base – instance of *c_Base*
- sizes – the configuration indicators: *dp_hasIndexFileSize and dp_hasDataFileSize* (their sum)
- the index cache – the configuration indicator: *dp_hasIndexCache*
- the linked managed element: the instance of the *c_Base*
- rule purpose – *i_SelfHealing* (as the minimum condition must be fulfilled).

The hard scenario is similar to the easy scenario with one big difference which renders it much more complicated: the existing ontology concepts are not sufficient to express the new pieces of information. This requires modification of the DSS ontology model, therefore the third step of the previous process is replaced by:

- Identify the non existent concepts and add them to the ontology, while ensuring that the new model is consistent
- Create the rule body using the (new) existing ontology concepts

This process has a double risk; this is why more validation is in order with modifying the initial ontology model. The problem comes from the fact that modifying an existent model implies a possible modification to the output of the already existing rules. Still, in the logic of the approach, the construction is incremental and disjoint. If a good generic model is given in start, any added concepts won't interfere with the existing rule base (or the impact is small enough to be controlled). Consider the same example as above, only in this case the notion of index cache value is unknown to the DSS model. The process becomes:

- Identify the existing correspondences wit the 'old' model
- Essbase Base – instance of *c_Base*
- sizes – the configuration indicators: *dp_hasIndexFileSize and dp_hasDataFileSize* (their sum)
- the linked managed element: an instance of the *c_Base*
- rule purpose – *i_SelfHealing*

- Create a new datatype property *dp_hasIndexCache* for the indentified concept (instance of *c_Base*).

7.4.5.2.2 Rule impact

Once the rule translation is made, the second aspect is assessing the impact of the new rules with the existing knowledge base. For this operation the requirements are: (i) to ensure that the rule has not already been declared, (ii) to have total control over the rule output and its implications with the functioning of the existing rules and (iii) to ensure that the new rule doesn't render the model inconsistent. We take each of them in order.

Rule duplicates. Identify whether the rule has already been declared. This is done when adding the rule, by detecting identical individuals if the rule body is specified directly in the data type property. If the rule body is specified within a file, it is harder to check, and the operation is done manually, so it is falls to the human expert. If the rule passes for various reasons, there is one last check that can be done to ensure the duplication. The test concerns the condition and the output of the rule. This means comparing the match demanded triplets (as the order may change) and the output triplets.

Rule model consistency ensures that the result of the rule doesn't render the model inconsistent. This is done with the help of a reasoner, once the rule is executed. For example, consider a rule that changes the value of the index cache. Due to a mistake of types, the rule sets the property a string value instead of a double value ("200"^^xsd:string in the place of "200"^^xsd:double). After the modification has been made, the model becomes inconsistent.

Rule interaction. Once the rule is accepted into the system (it is not a duplicated rule and it keeps the model consistent), the last part is to identify its interaction with the other rules from the system. This falls to the shoulders of the human expert. It is true that with a big number of rules this becomes harder to follow, but as seen from the state of the art , it is one of the flaws of the system. The maximum that can be done here is passing through a series of testing procedures to ensure that the new rule system works as intended.

Example. There is an existent rule that states: *[Rule 1] the minimum index cache for an Essbase base is 3MB*. On the technical forum, a new piece of information (not yet integrated in the system) states that: *[Rule 2] for data sizes grater than 1GB the minimum index cache size is 10MB*. These two rules translate into ontology rules:

```
[Rule 1- already existing in the KB
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
        ?base cp:dp_hasIndexCacheMin "3"^^xsd:double.
}
WHERE
{
        ?base rdf:type cp:c_Base.
}]
```

*[**Rule 2- new candidate for the KB***
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
?base cp:dp_hasIndexCacheMin "10"^^xsd:double.
}
WHERE
{
?base rdf:type cp:c_Base.
?base cd:dp_hasIndexFileSize ?ifs.
?base cd:dp_hasDataFileSize ?dfs.
FILTER (?ifs + ?dfs > "1000"^^xsd:double)
}]

When adding Rule 2, the expert knows of the existence of the first rule. If both rules are integrated in their current form, there will be an inconsistency in the model as the *dp_hasIndexCacheMin* property is functional and it will have two values for a base over 1Gb: "3" and "10". To solve this, the expert modifies Rule 1 by making it disjoint with the second rule. The complementary condition below is added to Rule 1, allowing the integration of both rules:

?base cd:dp_hasIndexFileSize ?ifs.
?base cd:dp_hasDataFileSize ?dfs.
FILTER (?ifs + ?dfs <= "1000"^^xsd:double)

7.4.5.3 Autonomic computing dynamics over the ontology model

We describe in this subsection the temporal dynamics of the ontology model. The state of the art showed that temporal dynamics is a particular sensible subject with ontologies. The AC dynamics ontology expression is strictly related to the order of executing the four phases of the ACM loops.

Retaking the diagrams from the UML model, we recall that there were several criteria that enabled the dynamics of the system: the order of the CHOP principles, the ACM MAPE-K loop phases for the managed elements, the MAPE-K loops for higher ACM levels etc. The dynamics expressed by these criteria relies on the changing of states of the managed elements, with instances from the *c_States* class. There is a state dependency for the symptom-diagnostic-heal cycle (SDH).

The ontology model allows the description of these passages between the states, but it should always be remembered that the actual actions or heals are performed via the effectors. The ontology model stops at the suggested actions and obviously cannot perform them. This is why the AC dynamics is expressed as a combination between the ontology model (with the rules) and external factors that permit action taking (i.e. 3rd party software, human intervention etc.).

To illustrate the ontology rule dynamics we first present a complete process of the MAPE-K loop for a Base over the Self Healing principle; then, we proceed with the ontology rule description of the two heuristics presented in the UML modeling.

7.4.5.3.1 MAPE-K loop process example

The purpose of this example is to express a full MAPE-K loop with the ontology model. We illustrate the entire process, starting from the monitored BI Copilot data and ending with the effectors that execute the required actions.

We present the expression of the following problem: For an Essbase base, in function of its size, the data fragmentation levels influence its performances. A base export, reset and reload are required when performances are under the specified levels. Table 31 expresses the information to be integrated with the ontology model:

Table 31 : Base defragmentation analysis

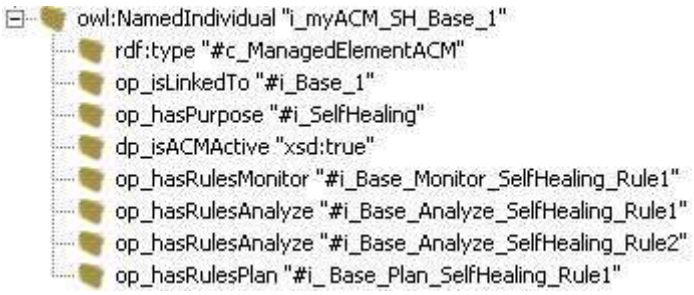
Performance requirement	**Base access mode**	**Base size (MB)**	**Fragmentation Quotient**	**Action**
80% of the queries < 1s	Buffered / Direct IO	0 - 200	>= 60%	Export, reset reload and the base
	Buffered / Direct IO	200 – 2000	>= 40%	
	Buffered / Direct IO	>= 2000	>= 30%	
	Direct IO	<= 50	>= 75%	

First, the BI Copilot DBs play the role of the sensors for the base ACMs. The data is loaded into the ontology via the DB to OWL transformation, for each of the bases, resulting in:

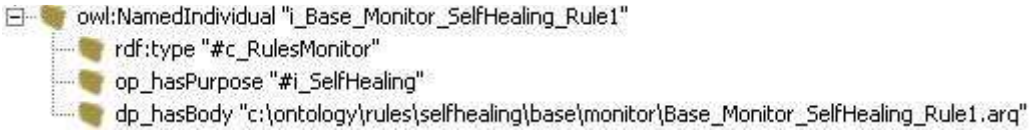
Table 32 : Base defragmentation data example in the loading DB to OWL phase

c_Base	**dp_hasAvg 80QRT**	**dp_hasAccesss Mode**	**dp_hasData FileSize**	**dp_hasIndex FileSize**	**dp_hasFragmen tationQuotient**
i_Base_1	“1.25”^^ xsd:double	“Buffered”^^ xsd:string	“700”^^ xsd:double	“100”^^ xsd:double	“0,55”^^ xsd:double
i_Base_2	“0.75”^^ xsd:double	“Direct”^^ xsd:string	“2000”^^ xsd:double	“1000”^^ xsd:double	“0,65”^^ xsd:double
i_Base_3	“1.9”^^ xsd:double	“Direct”^^ xsd:string	“25”^^ xsd:double	“5”^^ xsd:double	“0,82”^^ xsd:double

Second, we can proceed with the actual MAPE-K loop and its phases. The description of the loop is done through the properties of the instance of the base ACM:



Where the four rules are described as instances of the *c_Rules* class and have their bodies declared into text files:



Monitor. We already have almost all the data needed from the previous table. Almost, because there is a configuration indicator required by the problem which is not part of the retrieved data: the total size of the base. We have the base data file size and the base index file size, and need to compute their sum*)*, with the rule below (*Base_Monitor_SelfHealing_Rule1.arq):*

File: Base_Monitor_SelfHealing_Rule1.arq

```
[Rule 1-computes the base size
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
    ?base cp:dp_hasSize ?base_size.
}
WHERE
{
      ?base rdf:type cp:c_Base. // it is a base, so the base ACM
      ?base cp:dp_hasIndexFileSize ?ifs.
      ?base cp:dp_hasDataFileSize ?dfs.
      LET (?base_size := ?ifs + ?dfs).
}]
```

Analyze. Having now all the elements for the analysis, the ACM passes to the analysis, by describing two rules: the first determines whether or not the system needs healing (the performance condition) and the second determinates whether or not the fragmentation ratio is good.

The performance condition rule assigns an error diagnostic of high QRT performances for all concerned bases:

File: Base_Analyze_SelfHealing_Rule1.arq

```
[Rule 1-decides whether or not the performances are a problem
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
    ?base cp:op_hasDiagnostic cp:i_Error_Performances_HighQRT.
}
WHERE
{
      ?base rdf:type cp:c_Base.
      ?base cp:dp_hasAvg80QRT ?qrt80.
      FILTER (?qrt80 > "1"^^xsd:double)
}]
```

Once the rule is executed, *i_Base_1* and *i_Base_3* are affected and have the *i_Error_Performances_HighQRT* diagnostic for their *op_hasDiagnostic* property.

The fragmentation ratio rule adds the *i_HighFragmentationRate* diagnostic to the concerned bases:

File: Base_Analyze_SelfHealing_Rule2.arq

```
[Rule 1-decides if the fragmentation ratio is good or not
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
   ?base cp:op_hasDiagnostic cp:i_HighFragmentationRate.
}
WHERE
{
      ?base rdf:type cp:c_Base.
      ?base cp:dp_hasFragmentationQuotient ?fq.
      ?base cp:dp_hasSize ?size.
      ?base cp:dp_hasAccessMode ?am).
      FILTER (
      ?size > 0 && ?size <= 200 && ?fq >= 60) ||
      (?size > 200 && ?size < 2000 && ?fq >= 40) ||
      (?size >= 2000 && ?fq >= 30) ||
      (?size <= 50 && ?fq >= 75 && ?am = "Direct") )
}]
```

After execution, all three bases have a high fragmentation rate diagnostic. The ***op_hasHeal*** property of the ***i_HighFragementationRate*** diagnostic links with the actions: ***i_BaseExport, i_BaseReset and i_BaseReload***. The next step is to determine whether or not these actions are planned and executed.

Plan. On one hand, there is a high fragmentation ratio for all of the three bases. On the other hand, a high QRT error diagnostic is specified only for two of the bases. Therefore, the planning rule decides which actions will be performed:

File: Base_Plan_SelfHealing_Rule1.arq

```
[Rule 1-decides if the actions will be planned
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
      ?base cp:op_hasHeal ?heals_1.
      ?base cp:op_hasHeal ?heals_2.
}
WHERE
{
      ?base rdf:type cp:c_Base.
      ?base cp:op_hasDiagnostic cp:i_HighFragmentationRate.
      ?base cp:op_hasDiagnostic cp:i_Error_Performances_HighQRT.
      cp: i_HighFragmentationRate cp:op_hasHeal ?heals_1.
      cp:i_Error_Performances_HighQRT cp:op_hasHeal ?heals_2.
}
```

*Only if the two diagnostics are present to*gether then heals will be performed. *i_Base_1* and *i_Base_3* have the heals of *i_BaseExport, i_BaseReset, i_BaseReload* (from the *i_HighFragementationRate)* and *i_ActionsNA* (from the *i_Error_Performances_HighQRT*). The latter action indicates a generic action, as a high QRT diagnostic by it self disposes of no specific heal.

Execute. The required action concepts are linked in the ontology with machine/human actions (i.e. scripts, programs etc.) This is where the ontology model "ends". The planned actions are further on interpreted by a software program that will be in charge of their execution. Therefore, the execution phase of the ACM ontology model doesn't exist in its purpose, the ontology will not perform the specified actions by itself. It exists on its declaration, as due to the rule chaining and the expression of the heals concepts, we are able to determine which actions are executed for which base.

From this example we proceed to the expression of the dynamics through the two heuristics, which details the ontology temporal evolution of indicators.

7.4.5.3.2 Self optimization algorithm

The expression of this heuristic in the ontology model introduces the notion of temporal 'pervious' configuration/performance indicators. We retrace the same steps as in the rule example above for understanding the ontology modeling of the heuristic.

The parameters of the self optimization heuristic are instances of the *c_Algorithms* class. These parameters are not present in the BI Copilot DBs, and their values are manually filled by the human expert before launching the self optimization ACM.

Table 33 : Self Optimization algorithm parameters example

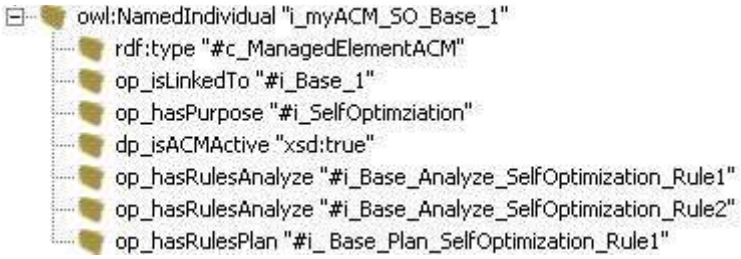
c_Algorithms	dp_hasDelta	dp_hasStep
i_Algorithm_1	"0.05"^^xsd:double	"0.4"^^xsd:double

First, the BI Copilot DBs offer the configuration and performance indicators (exemplified in Table 34).

Table 34 : Self Optimization ontology indicators example

c_Base	dp_hasAvg80 QRT (xsd:double)	dp_hasAvg80 QRTPrev (xsd:double)	dp_hasIndex Cache (xsd:double)	dp_hasIndex CachePrev (xsd:double)
i_Base_1	1.25	1.1	50	60
i_Base_2	0.75	0.9	100	85
i_Base_3	1.9	1.5	3	3.2

We notice the introduction of the two "*prev" properties, which indicate the previous values of the indicator, enabling a first temporal axis. Whenever a past time stamp is required, the concerning properties are 'duplicated' in the ontology model: "*prev" becomes "prev1day", "prev2weeks", "prev1month" etc. As it is implemented by the base self optimization ACM, the instance of the *c_ACM* class looks like:



The ACM has no *monitor* rules for this heuristic in the ontology model. The first executed rule is the *analyze* rule which decides whether or not the previous cache changes violate the performance threshold:

File: Base_Analyze_SelfOptimization_Rule1.arq

```
[Rule 1-decides wether the old cache change was valid from the QRT perspective
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
    ?base cp:op_hasDiagnostic cp:i_ValidCacheChangePrev.
}
WHERE
{
      ?base rdf:type cp:c_Base.
      ?base cp:dp_hasAvg80QRT ?qrt.
      ?base cp:dp_hasAvg80QRTPrev ?qrt_prev.
      cp:i_Algorithm_1 cp:dp_hasDelta ?delta.
      FILTER ((?qrt <= "1"^^xsd:double) &&  // keep the initial constraint valide
      ((?qrt <= ?qrt_prev) ||  // either the cache change didn't affect the performances
      (?qrt > ?qrt_prev && ((?qrt-?qrt_prev)/?qrt < ?delta) )) // the performance drop
is within the limits
}]
```

The second analyze rule is based on the valid cache change diagnostic, and attempts to lower further the cache values. The new cache is computed further with the rule:

File: Base_Analyze_SelfOptimization_Rule2.arq

```
[Rule 2-attempts a new cache change
PREFIX cp:<http://localhost/CacheProtoBC.owl#>
DELETE
{
    ?base cp:dp_hasIndexCachePrev ?cach_prev. //delete the old cache value
}
INSERT
{
      ?base cp:op_ hasIndexCachePrev ?cache.
      ?base cp:op_hasIndexCache ?new_cache.
      ?base cp:op_hasDiagnostic cp:i_ValidChangeCache. // go on and plan the cache
change
}
WHERE
{
```

```
        ?base rdf:type cp:c_Base.
        ?base cp:op_hasDiagnostic cp:i_ValidCacheChangePrev.
        ?base cp:dp_hasIndexCache ?cache.
        ?base cp:dp_hasIndexCacheMin ?min_cache.
        cp:i_Algorithm_1 cp:dp_hasStep ?step.
        LET (?new_cache := ?cache - ?cache * ?step). // compute the new index cache
        FILTER (?new_cache >= ?min_cache) // while keeping the minimum levels
}]
```

The second rule is dependant on the first rule over *i_ValidCacheChangePrev*. If the rule is satisfied then the *i_ValidCacheChange* diagnostic is set indicating to the planning phase that a cache change needs to be performed.

Plan. The planning rule indicates which healing actions are executed. In this case, the *i_ModifyCaches* heal implies three actions: stop the application, mirror the ontology values to the DSS configuration and restart the application.

File: Base_Plan_SelfOptimization_Rule1.arq

```
[Rule 1-plans the heals actions to be executed
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
    ?base cp:op_hasHeal ?heal.
}
WHERE
{
    ?base rdf:type cp:c_Base.
    ?base cp:op_hasDiagnostic ?diag.
    ?diag cp:op_hasHeal ?heal.
}
```

Execute. The ontology can't act on its own so the actions are performed by the effectors. In this case, the effectors are software programs which use to a series of Essbase scripts that modify the cache values.

7.4.5.3.3 Self configuration heuristic

The self configuration heuristic ontology modeling illustrates how the managed element ACMs communicate, as it requires exchanges between the managed element levels. It is based on the utilization periods defined in the ontology, the physical server resources and on the managed elements performances. We remind that the SLA utilization periods are not specified in the BI Copilot DBs, but directly asserted in the ontology according to client specifications (Table 35)

Table 35 : Self Configuration ontology periods example

c_Physical Application	c_Calendar Specific	dp_hasCalendar Period	dp_hasCalendar IntervalValue	dp_hasCalendar Day
i_APP_1	i_Period_1	"M"^^xsd:string	1,2,3^^xsd:int	1,2,3,4,5^^xsd:int
i_APP_2	i_Period_2	"M"^^xsd:string	7,8^^xsd:int	15^^xsd:int

The managed elements resources and performances are obtained similarly with the previous example, from the BI Copilot DBs.

The dynamics of this heuristic is reinforced by the fact that it combines different type managed elements. The MAPE-K loops chain differently as ACMs require information from other ACMs. Therefore, the loops are executed several times to ensure the passage. In our modeling, the loops are executed in accordance with the DSS hierarchy, and this is how we further present them.

MAPE-K Loop Base ACM

Is the first executed loop. The rules for the base ACMs try to see if the application has freed some memory and allocated them a part of it, in order for the ACM to perform a cache reallocation.

Monitor – has one rule that computes the free memory for each base as a difference between the base allocated and occupied memory. The base allocated memory is changed by the physical application ACM, and the base occupied memory is the sum of its caches. The fact that the base allocated memory is controlled by the physical application ACM implies that the base loop will be executed at least two times (to ensure at least one passage through the physical application ACM loop and one cache change execution).

File: Base_Monitor_SelfConfiguration_Rule1.arq

```
[Rule 1-computes the base free memory
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
        ?base cp:dp_hasFreeMemory ?fm.
}
WHERE
{
    ?base rdf:type cp:c_Base.
    ?base cp:dp_hasOccupiedMemory ?ocm. // taken from the BI Copilot DBs
    ?base cp:dp_hasAllocatedMemory ?am. // property changed by the physical application ACM. //At first loop passage this property is either empty or contains a non updated value
    LET (?fm := ?am - ?ocm).
}]
```

Analyze the rules look at the base free memory and, in function of its value, raise an error diagnostic (if negative), or a cache reallocation diagnostic (if grater than 0):

File: Base_Analyze_SelfConfiguration_Rule.arq

```
[Rule 1-if free memory is negative then set an error diagnostic
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
    ?base cp:op_hasDiagnostic cp:i_Error_NotEnoughMem.
}
```

```
WHERE
{
      ?base rdf:type cp:c_Base.
      ?base cp:dp_hasFreeMemory ?fm.
      FILTER (?fm < 0). // base uses more than it has allocated
}]
```

```
[Rule 2-if free memory is positive then set a base optimization diagnostic
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
   ?base cp:op_hasDiagnostic cp:i_NonOptimalCacheConfiguration.
}
WHERE
{
      ?base rdf:type cp:c_Base.
      ?base cp:dp_hasFreeMemory ?fm.
      FILTER (?fm > 0). // there is some free memory available for this base
}]
```

The ***planning*** and ***execution*** phase are similar to the previous example. The planning recovers the heals from the diagnostics of the managed element, and the execution triggers the changes required by these heals.

MAPE-K Loop Physical Application ACM.

The second ACM, which runs its loop after the base ACM, is in charge of allocating the available memory between the bases of the same application. It is dependant on the logical server ACM, thus similar to the base ACM it will run at least 2 times in order to obtain the amount of memory it can allocate.

Monitor the monitor rule is similar to the base monitor rule, as it computes the available free memory to be allocated to the bases. Also, the physical application modifies the performance levels according to the utilization period, by scaling the QRT into the QoS.

File: PhysicalApplication_Monitor_SelfConfiguration_Rule1.arq

```
[Rule 1-computes the physical application free memory
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
   ?app cp:dp_hasFreeMemory ?fm.
}
WHERE
{
      ?app rdf:type cp:c_PhysicalApplication.
      ? app cp:dp_hasOccupiedMemory ?ocm. // taken from the BI Copilot DBs
      ? app cp:dp_hasAllocatedMemory ?am. // property changed by the logical server
      ACM. //At first loop passage this property is either empty or contains a non
      updated value
      LET (?fm := ?am - ?ocm).
}]
```

File: PhysicalApplication_Monitor_SelfConfiguration_Rule2.arq

```
[Rule 2-modifies the QoS with the utilization periods
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
INSERT
{
    ?app cp:dp_hasQoS ?qos.
}
WHERE
{
      ?app rdf:type cp:c_PhysicalApplication.
      ?app cp:dp_hasAvgQRT ?avg_qrt.
      ?app cp:op_hasUtilisationPeriod ?up.
      ?up cp:dp_hasCalendarYear ?up_year.
      ?up cp:dp_hasCalendarDay ?up_day.
      ?up cp:dp_hasCalendarInterval ?up_int.
      ?up cp:dp_hasCalendarIntervalValue ?up_intval.
      cp:i_CurrentDate cp:dp_hasCalendarYear ?cd_year.
      cp:i_CurrentDate cp:dp_hasCalendarDay ?cd_day.
      cp:i_CurrentDate cp:dp_hasCalendarInterval ?cd_int.
      cp:i_CurrentDate cp:dp_hasCalendarIntervalValue ?cd_intval.
      {// application in utilization period
      LET (?qos := ?avg_qrt). //the QoS is the QRT if application in utilization period
      FILTER (?up_year = ?cd_year && ?up_day = ?cd_day && ?up_int = ?cd_int
      && ?up_intval = ?cd_intval)}
      UNION
      {//application outside the utlisatino period
      LET (?qos := (?avg_qrt / 4)). //the QoS is the QRT divided by 4
      FILTER (?up_year != ?cd_year || ?up_day != ?cd_day || ?up_int != ?cd_int ||
      ?up_intval != ?cd_intval)}
}]
```

Analyze: allocates the memory to the low performance bases. First, the rules determine their number and second, they redistribute the free memory between them. The used performance indicator is the QoS:

File: PhysicalApplication_Analyze_SelfConfiguration_Rule1.arq

```
[Rule 1-allocates the free memory to the non performing bases
PREFIX cp: <http://localhost/CacheProtoBC.owl#>
DELETE
{
    ?base cp:dp_hasAllocatedMemory ?am. // removes the old value
}
INSERT
{
      ?base cp:dp_hasAllocatedMemory ?new_am.// the modification needed by the
      base ACM
      ?app cp:op_hasDiagnostic cp:i_NonOptimalCacheConfiguration. // thus
      requiring a cache //reconfiguration with the new memory values
}
WHERE
{
```

?app rdf:type cp:c_PhysicalApplication.
?base rdf:type cp:c_Base.
?app cp:op_hasChild ?base.
?app cp:dp_hasQoS ?app_qos.
?base cp:dp_hasQoS ?base_qos.
?base cp :dp_hasOccupiedMemory ?occ_am.
?app cp:dp_hasFreeMemory ?app_fm.
?app cp:dp_hasNumberOfNonPerformantBases ?npb.
FILTER (?app_qos < ?base_qos). //non performing bases
LET (?new_am := ?occ_am + ?app_fm / ?npb) //compute the new memory
}]

Plan & Execute the cache reconfigurations with the new values are done in similar ways as presented previous.

MAPE-K Loop Logical Server ACM / MAPE-K Loop Physical Server ACM

The two ACMs will execute in this order and retake the same considerations from the physical application ACM, for the memory reallocation rules.

Summing up the ontology rule adoption, the dynamics of the self configuration requires the following passage of the loops twice over the DSS hierarchy:

Base MAPE-K – Physical Application MAPE-K – Logical Server MAPE-K – Physical Server MAPE-K – MAPE-K – Physical Application MAPE-K – Logical Server MAPE-K – Physical Server MAPE-K

7.5 Conclusion

The section presented our proposed approach with this thesis. It was split into two main parts: the proposed UML modeling and the proposed ontology modeling.

First, the UML model presented our vision over a complete DSS autonomic management model, with two fundamental aspects: static and dynamic. The static aspect concerned the hierarchical organization of the DSS with its configuration and performance parameters. The dynamic model included the description of the autonomic computing adoption for DSS management. It described the AC hierarchical model, with the organization of the ACM levels and the MAPE-K loop. Moreover, the dynamic knowledge integrated the non-structured pieces of information with the management model, with the help of formalization rules organized by criteria such as managed element hierarchy or AC principles and MAPE-K loop phases.

Second, before the description of the ontology model, a series of UML-DB/OWL translations have been presented, with the purpose of understanding the construction of the ontology model. The information is formalized with both relational DBs (BI Copilot) and with ontologies (BI Self-X). The translation considerations were based on previous works which detailed the subject.

Third, we have presented the ontology model, with all the information presented din the UML modeling plus some extra information derived from the inference. We have seen

that the static aspect has already been implemented by SP2 with the BI Copilot DBs, and that the ontology model completed the dynamic aspect gap, with the usage of inference engines and ontology based rules. We have presented the process of loading a non-structured piece of information into the ontology, and exemplified the AC dynamics and ACM communication via the two self management heuristics.

As a sum up, our proposition tried to build a valid and viable DSS management knowledge model. The question: why ontologies and not relational bases doesn't really have sense in this context, as they are both used. All the powerful mathematical support and aggregated information offered by the ACMDBs alongside with the temporal dynamics is used by the ontology model as a constant information source. The ontology information depends and is populated with the data from the DBs. The semantic BI Self-X knowledge bases and the BI Copilot DBs coupled together form our complete DSS management solution.

8 Experiments and Results

"Defect-free software does not exist."

W. Venema

8.1 Introduction

The experimental environment and the results we have obtained provide the arguments for and against our approach. Because this is a new application domain, previous comparison results don't exist in most of the experiments we have conducted. Nevertheless, we try to show similar results where the case. The most notable result that we have obtained is the important reduction in the time spent by human DSS experts to manage the data warehouses. This corresponded to the initial purpose of our approach: help humans better manage their DSS. The level of effectiveness is discussed with the results.

We present first the software products that have been developed with the approach, and the experimental prototype (BI Self-X). We present the general component architecture and the functioning and implementation details of each component. Afterwards, we proceed to the description of the experimental environments, by separating the 'laboratory' theoretical environments from the real client environments. There are different methodologies and protocols used in each case, and we detail them in turn. This environment separation influences in the next section the performed experiments, as we present our results under 'perfect' conditions and under real client conditions. Each obtained result is discussed, to asses its gain (if any) and to compare it with other methods (if existent). We close the chapter with the conclusions, and a note on how our results impact actual data warehouses management and the future of DSS.

8.2 Software products – experimental prototype

This section presents the software implementation of our approach, BI Self-X, with its integration with the other SP2 software solutions. We first describe the general architecture of BI Self-X, and then we go into the technical details of each module.

8.2.1 General architecture

The general architecture shows how the BI Self-X software module is integrated with the rest of the SP2 solutions. The BI Self-X is a separate module, and, it contains in turn three sub modules organized by their functionality (Figure 81).

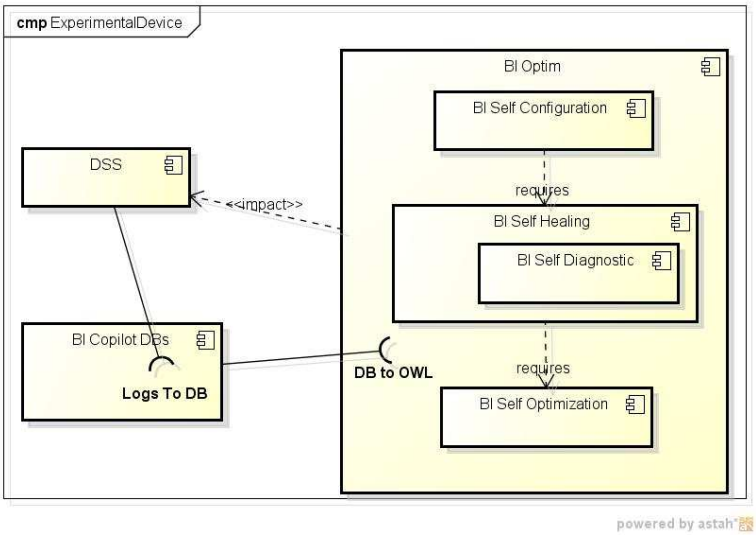


Figure 81 : BI Self-X experimental device integration

First, the DSS is a separate component in its whole. Then, the BI Copilot DBs, build from the logs and traces of the DSS activity, offer an interface to the specific data warehouse implementation (Oracle, SAP, IBM etc.). The two components are external to the component which implements our approach, BI Self-X. Nevertheless, any implementation of BI Self-X requires the existence of a data warehouse architecture and its corresponding BI Copilot DB model. It also needs the interface that translates the data from the BI Copilot DBs, and populates the ontology with the respective information. The BI Self-X module contains three sub modules, which correspond to the first three principles of AC: self configuration, self healing and self optimization. Each of the modules is dependant on the previous one in the CHOP order.

- *Self Configuration module*: assures the first level of AC adoption, keeping the configuration of the data warehouses within the working limits and in conformity with the best practices.
- *Self Healing module* – makes sure that the DSS has little (or no) errors while functioning. The implementation of the module assures two functionalities (with two sub modules):
    - *Self Diagnostic* – alerts the human expert over the causes of error and the various diagnostics of the DSS. It equally proposes a series of healing actions, without executing them.
    - *Self-Healing Action* **–** uses the diagnostics healing actions, to perform them in an autonomic manner without requiring human intervention.
- *Self Optimization module* - requires passage through the self configuration and the self healing module, and implements t improvement and optimization capabilities. The usage of this module can either be
    - In pre-production, when new configurations are required. This is done via loading and stress tests to find suitable (if not optimal) starting production configurations.

- While in production, if the situation allows it, the self optimization module improves the resource/performance ratio, without impacting the DSS over the specified limits.

Note. We note the absence of the Self Protection module. This is due to the fact the focus with this thesis is on the first three principles only. We have mentioned this principle both in the state of the art and the models, but the experimental prototype doesn't implement it.

Regarding the physical implementation, the DSS can gather several physical machines, as data warehouse hosts. The interface which builds the BI Copilot DBs is on another machine, has connections with the data warehouse physical servers (ftp, http etc.) and is able to gather all the traced data. Then, the constructed DBs can be stored on a separate machine. Finally, the BI Self-X module can be deployed on a third machine (with all its sub modules and the DBs to OWL interface). Of course, they can be together on the same machine, but this depends on the chosen implementation architecture by the clients, who sometimes are reticent when it comes to installing now software products on existing configurations. There is a high flexibility with the deployment, such that even all the analysis part (BI Copilot DBs + BI Self-X) can be done on SP2 machines, starting from the logs extracted from the clients DSS servers.

The implementation of the ontology model is done inside BI Self-X, as the software product modifies directly the ontologies via the API interface. BI Self-X is built in such a manner that enables autonomy up to the specified level. Moreover, it can directly modify the data warehouse configurations or leave the actions to be performed by the human expert (e.g. a non trusty client can demand that all the proposed actions are only proposed to the human experts, without any specific autonomic action taken by the software).

In consideration with the general implementation architecture, there are several elements that need to be discussed, specifically: solution reproducibility, implementation generics and simulation considerations.

Reproducibility **-** a first question that is posed when implementing an experimental prototype is if it is reproducible, if the experiences and the deployment of the software are easy and scalable and if the same results are obtained under similar conditions. We have tested the BI Self-X prototype for Essbase systems, for us and some of our clients. The self-configuration and self-healing modules function under the government of the same rules, thus the changes and diagnostics are the same, independent of the DSS. The question was posed when implementing the optimization algorithms, as here the conditions of usage change from system to system. Yet, as we shall see with the experiments, the results all show the same tendency in improving performances.

One very important aspect of reproducibility is scale passing. Prototypes are usually used for small scale experiments and under laboratory environments. High load charges and results obtained with benchmarks such as [tpc (2010)] can offer a good view of the systems reproducibility. Unfortunately, our approach was not suitable for testing under these environments. We have successfully proven a scale passage of the prototype, in terms of a higher number of data warehouses over a higher activity. Reproducing BI Self-X under

various conditions also implies the reproducibility of the BI Copilot architecture for the given environment. The two are coupled together, and BI Self-X depends on BI Copilot. The degree of liberty offered by the latter impacts directly the one for our approach.

One last aspect of reproducibility concerns the 'noise reduction'. Even under laboratory environments, there will never be two perfectly identical experiments. We always use an OS as deployment platform and we don't control every way in which it uses its memory, disks or CPUs. Fluctuations in the performance occur all the time. For example, our experiments have shown that under identical cache configurations for the same data warehouses, over the same activity, the query response time has an average variation of up to 10%. These variations are expressed as 'noise', and must be taken into consideration when speaking of reproducibility.

Generics – the generics of the prototype refers to two different aspects: (i) platform independence and (ii) data warehouse implementation generics.

Platform independence means that the BI Self-X software can be deployed on different OSs. As it is completely implemented with Java, the cross platform compatibilities are not an issue. The problems may appear at the interconnection with the other SP2 modules, as the technologies behind them are platform related (.NET for the interface with the BI Copilot module and SQL Server DBs for the BI Copilot DBs). But, as the architecture is modular, BI Self-X needs only a connection to the DBs, while the BI Copilot module is deployed on another machine. Therefore platform independence generics is assured.

The second aspect of *generics* refers to the data warehouse implementation. The software prototype, as well as the ontology model and the BI Copilot suite, are build for a specific data warehouse implementation (Oracle Hyperion Essbase). This problem is more accentuated with the DB model. We have tried to keep the ontology model as generic as possible (with the DSS architecture part of course - the AC adoption model remains the same no matter the DSS model). Nevertheless, some of the elements described don't always apply (e.g. the storage mode for Essbase Cubes (ASO & BSO) is not a valid measure for a BO cube; or the notion of physical application doesn't have a sense for Microsoft DSS solutions). Our view towards this problem is to differentiate between the generic and the specific notions, a task that implies a perfect knowledge over all the targeted systems. For now, the model mixes the two, assuring a first level of generics with which most of the DSS solutions can be fitted with.

An interesting discussion about generics refers to the data warehouse fundamental architecture (not the software implementation), with the Kimball / Inmon models. We have specified that our modeling suits the hierarchical Inmon proposition for data warehouses. Applying our techniques for a Kimball approach is possible at a first glance. As the model would contain only one managed level (no more managed elements hierarchies), the taxonomy considerations would no longer apply. Nevertheless, this would not render the model nor the implementation useless. We have not yet been in this situation, but we consider plausible an adaption of the BI Copilot DBs model and the BI Self-X ontology model for a Kimball data warehouse implementation. It can be one of the improvement points with our approach.

Simulations – Testing the prototype before putting it into production is a laborious task due to the existence of the multiple modules that must be assured and the complex environmental set up. This is why we had to use artifacts to simulate elements such as user activity (predefined and random), usage periods (day/night) or utilization periods (calendars). All this is explained in detail with the experimental environments.

8.2.2 Functioning

The overall functioning of BI Self-X is shown in Figure 82.

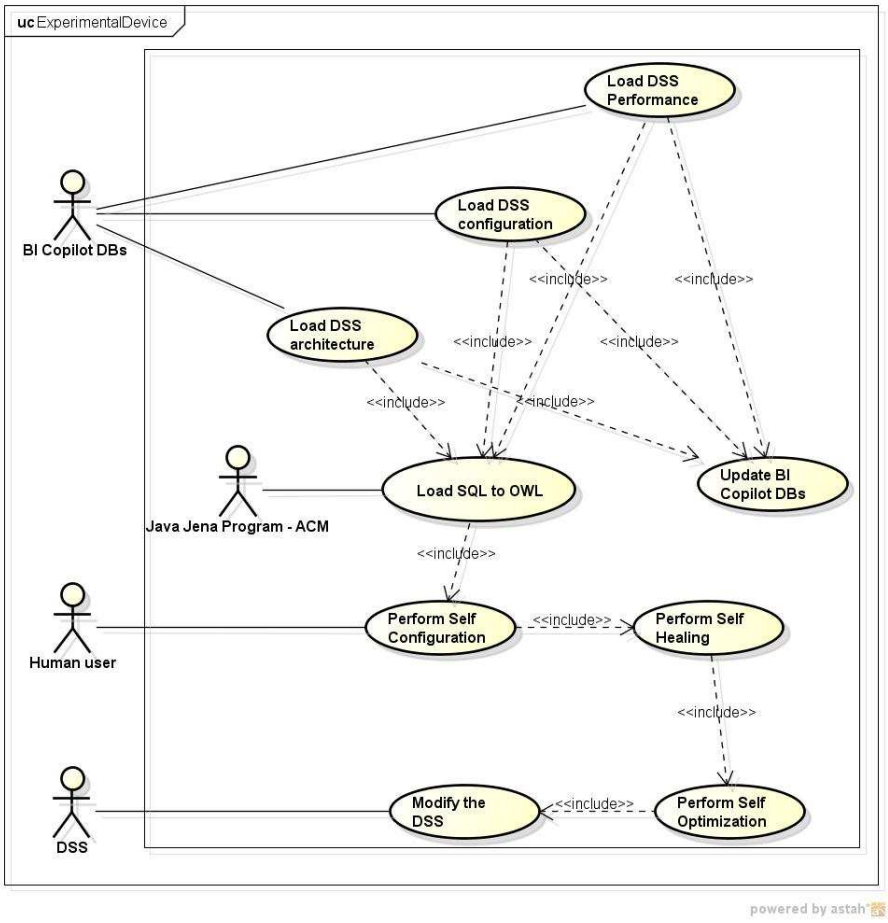


Figure 82 : Experimental Device functioning

The three main DSS architecture elements are also part of the functioning, with the first step of loading the data from the BI Copilot DBs to the ontology model. Each of the three elements, architecture, configuration and performance, is treated separately, and together they form the sub module that assures the DB2OWL interface. Then, depending on the use scenario and autonomic adoption level, each self-x module is executed, either with human intervention or in an autonomic manner. The diagram shows what happens over a single passage of the MAPE-K loops (for all the managed elements).

Execution timestamps are specific for each DSS. Self configuration and self healing can either be executed punctually a specific moment or at defined time intervals (e.g. each

hour, the self healing module activates the self healing ACMs, such that the system has a failure latency of 59 minutes). Self optimization can only be executed at regular defined time intervals (i.e. only one execution of the optimization ACMs loops doesn't bring much optimization; the more loops are executed, the better the optimization result is).

Evidently, the execution times depend on the updating intervals of the BI Copilot DBs. This is why, prior to loading the ontology, an update of these bases is requested. From the execution point of view, the BI Self-X processes wait the finishing of the BI Copilot processes (cf. with the use case).

8.2.3 BI Self-X implementation details

The software implementation details the BI Self-X prototype's technical elements and their interconnection. The BI Self-X module is completely written in Java (version 1.5 compatible), with the Jena API for ontology support, and the swing API for the GUI. The choices have already been argued, shortly because of the cross platform support and the big community around Jena. Figure 83 illustrates a detailed view of the components of the implementation:

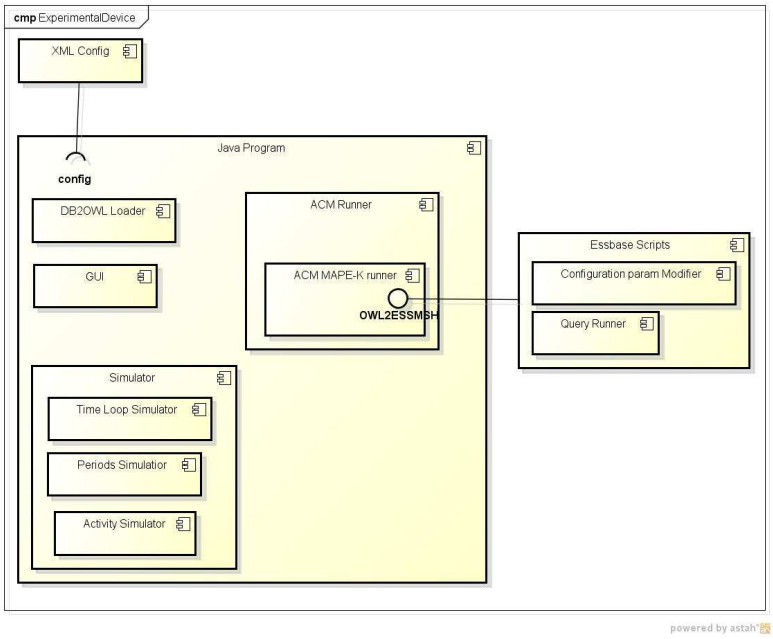


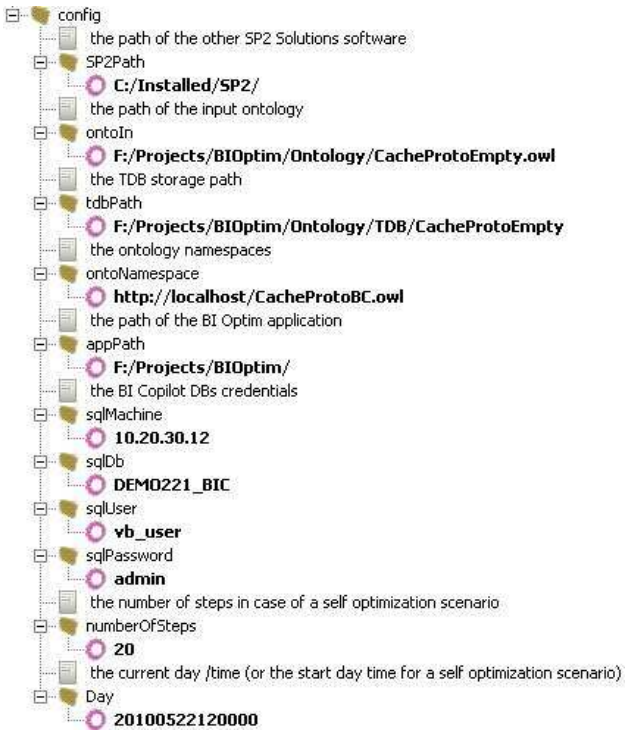
Figure 83 : BI Self-X software implementation details

XML Config – the BI Self-X configuration file; it is under the form of an xml document, specifying the following elements:

- The URIs of the ontology models and the paths to the ontology rules root folder
- The addresses of the BI Copilot DBs and the corresponding connection parameters, for the DB2OWL translator
- The date/timestamp for which a configuration or healing will be made (e.g. a self healing is required on the date of 25 April 2010 at 15:30:00 hours). For an interval

requirement, it equally specifies the ending date (e.g. optimization between the 25.05.2010 00:00:00 and 25.06.2010 00:00:00)

An example of an xml configuration file is shown below:



Java Software Program – is the main module, which implements the different sub modules of the software prototype. Among these we have:

- *A DB2OWL loader* that is in charge of loading the BI Copilot DBs into the ontology models, with DSS architecture, configuration and performance parameters. The loading is done with regards to the specified correspondences of the DB – OWL models.
- *A GUI* for communication with all the elements of the AC architecture. The role of the GUI is also to provide the manual manager interface. Moreover it provides the list of actions that are performed by the self configuration, the list of diagnostics and heals from self healing, and the evolution of parameters with the self optimization. The GUI permits the selection of the operation mode. There are several operation modes depending on the usage purpose of BI Self-X. It equally permits the configuration of the various parameters of 'during' execution, such as the active ACMs.
- *ACM Runner* is a module that simulates the chaining of the ACM loops for the managed elements. It can be seen as the orchestrated manager which coordinates the execution of the 'little' ACMs. As a part of the ACM runner we have the *ACM MAPE-K loop runner*, which is in charge of running a single loop for a single managed element only.
- *Simulator* is a module in charge of simulating the DSS activity. It has several sub-modules:

- *Time Simulator* – which basically changes the dates on the data warehouse machines so that it simulates daily intervals are simulated
- *Periods Simulator* – ensures the difference between the day and the night periods
- *Activity Simulator* – in charge of executing operations on the data warehouses, with the purpose of generating performance indicators (e.g. data reloading, cube recalculations, data query); it works either with predefined patterns or randomly.

The Java module is standalone, requires no installation or additional elements. All the required jar files are packed with the distribution, which includes:

- The JDOM library for xml interface
- The TDB libraries (including Jena) for the ontology interface
- The SQLJDBC driver library for the BI Copilot DB interface
- The default java libraries, including the swing libraries for GUI development

DSS Scripts (Essbase) – contains the scripts for accessing the specific data warehouse implementation (in our case Essbase). There are two types of scripts:

- *Configuration Modifier* – permitting configuration parameters modifications.
  Example. The following script modifies the access mode and the caches of an Essbase cube with the passed parameters ($1-$5):

```
login 'admin' 'password' on 'localhost';
    alter database $1 set io_access_mode $2;
    alter database $1 set index_cache_size $3;
    alter database $1 set data_file_cache_size $4;
    alter database $1 set data_cache_size $5;
logout;
```

- *Query Runner* – scripts that run various operations on the data warehouses such that human activity is simulated. It contains a pre-defined pool of operation queries.
  Example. A data retrieval query that returns all the products of type C, with all its sub products, on the Market A, for the second semester of each existing year, is shown below:

```
<Sym
<Supshare
<Column (Scenario, Year)
<CHILDREN Scenario
<IDESCENDANTS Qtr2
<Row (Product, Market, FY)
<IDESCENDANTS ProdC
<IDESCENDANTS MarketC
<CHILDREN FY
```

8.3 Experimental environments

The experimental environments refer to the full context in which the experiments are performed, from physical (machines, network etc.) to logical aspects (data warehouse sizes, user activity, SLA specifications etc.). We present several aspects that we take into consideration when we speak of an experimental environment, as follows:

- *The hardware architecture* –regarding the physical machines and their specifications, such as CPU speed, RAM or disk capacity. The hardware architecture describes also the links that exist between these machines (local networks, internet etc.) as well as any virtual representation of the physical servers (e.g. Microsoft Hyper V or VMWare ESX).
- *The logical software architecture* – concerns the software and communication protocols that are installed on the hardware architecture, such as: data warehouse architecture and configuration, operating systems, OS load and performance, network capacity etc. All the DSS architecture levels from the physical server level down concern the logical software architecture.
- *The user activity* **–** be it normal user, developer or DSS expert, the user activity is very important when performing an experiment. Different users have different usage patterns (e.g. developers perform long testing queries, normal users perform small fast queries etc.).
- *SLA/O s* **–** the levels of service and objectives are equal part of the experimental environment, and relate with the user activity. All the considerations regarding the day/night data warehouse usage purpose or the data warehouse utilization periods are taken into consideration as part of the experimental conditions.
- *Performance* **–** all aspects which indicate the performance, from the quality of service to the time needed for an expert to diagnose and 'heal' a DSS, are the indicators of the experiment validity.

We have split the experimental environments are into two main categories: (i) theoretical 'laboratory' environments and (ii) real 'client' environments. The considerations presented above are valid for both, but their implementations differ, as discussed in the following.

8.3.1 Laboratory environments

A laboratory (or theoretical) environment is specially created to maximize the performance gain of the experiment. It is performed by us and it is 'perfect' without any noise or interference. With the laboratory experiments we are able to control and isolate each module of the BI Self-X prototype. This is essential, as before going into production over real environments, the product must be tested several times for failures and bugs in the theoretical environment. This assures the overall quality standard of our solutions, and also provides the figures we can show our (future) clients.

The hardware architecture is build on top of a single physical machine that enables virtualization. This allows the simple reallocation of resources, specifically CPU power, RAM capacity and hard disk capacity, for each of the virtual machines (Figure 84):

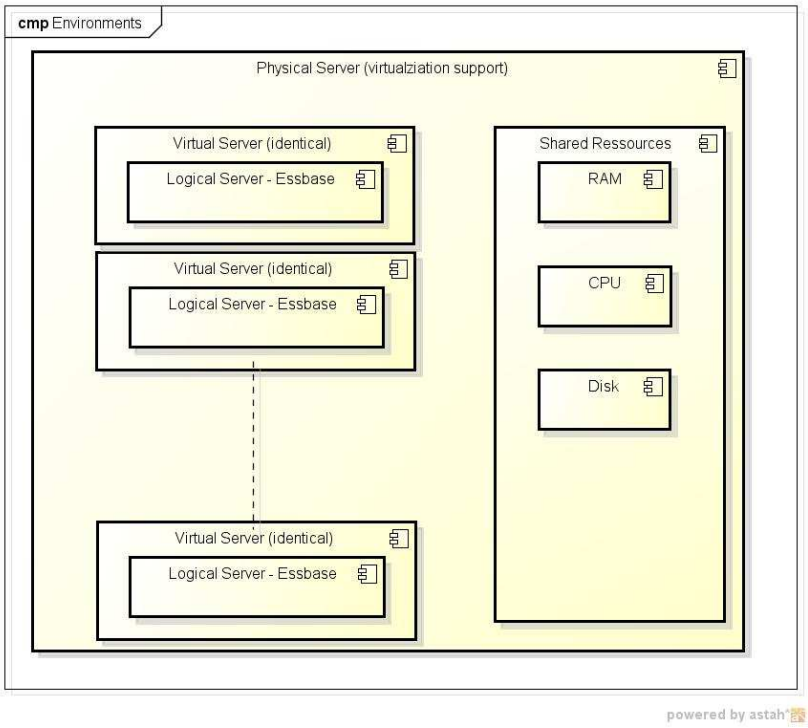


Figure 84 : Laboratory environment

The logical architecture includes identical logical servers on each of the virtual server. Even if the resources allocated for each virtual server differ, the logical server remains the same (e.g. Essbase version 9). The characteristics of the logical architecture are:

- Each virtual server contains only one logical server, the version of the logical server is the same
- Each logical server contains the same number of physical applications, with the applications being identical on all logical servers
- On a logical server, each physical application may contain a various number of bases. The application base configuration is duplicated on the other logical servers.
- Each base is the same, contains the same data and has the same structure. Therefore the data warehouse structure is a suite of mirrored identical applications (with identical bases).
- The parameter configuration of the bases differs, which allows us to isolate the performance indicators with the parameters, without taking into account the actual data.

The user activity is simulated via the simulator module presented earlier. As there are no real users, activity must be simulated. To that end we have created pools of predefined scripts aimed towards the generation of several performance indicators, as follows.

Retrieval / report generation time scripts - which perform data retrieval operations which are used to generate rapports: from simple (which take under a second) to high demanding (up to several minutes). Most of the QRTs revolve around the 1 second value (as observed from the real human activity). Figure 85 shows the report distribution for the query response time (under a good configuration):

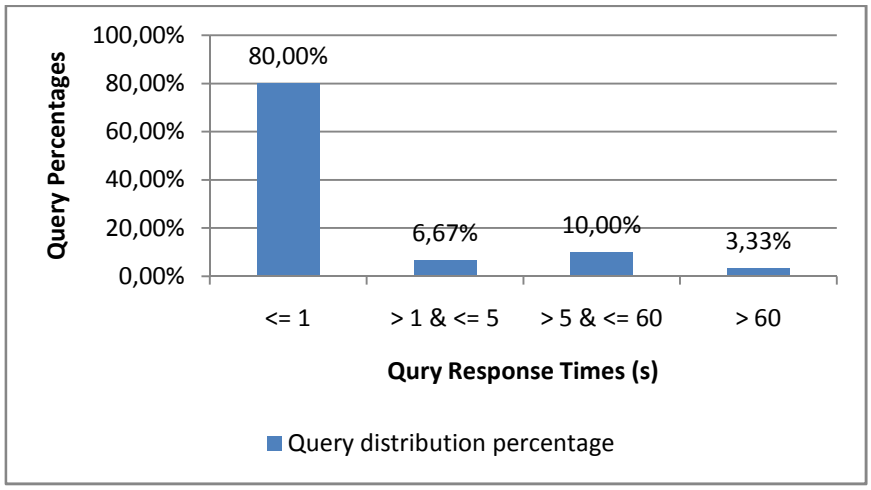


Figure 85 : Query response time distribution in a laboratory environment

Calculation times scripts –perform calculation operations such as sums or products. The considerations from the retrieval time are generally the same for calculation operations. In the laboratory environment, we provide the same a pool of calculation queries which are executed in the same manner every time (such that results are predictable).

Cube restructuration – demand re-computations of the aggregated data with data warehouses. Therefore it expresses as the time needed to reintegrate the new operational data into the aggregated dimensions. Depending on the resource allocation / parameter configuration, this operation can vary greatly from several minutes to several hours.

Data load – performance is measured by the times required to load data to the cubes (e.g. from plain files). A base exports, clears and reloads the data from the exported source. As we have the same bases, the data files are identical. The data load times differs on the parameter configuration and especially on the CPU power.

SLA/Os – service levels are defined by our DSS experts. The fact that this is a theoretical environment allows a higher flexibility. For instance, we can define different service levels for each base, whereas in a real environment they are usually defined by application or logical server. The service level agreements that we use concern solely the utilization periods and utilization purposes.

We define the utilization periods manually depending on the test, as well as the performance thresholds for each utilization period. They are specified in the ontology model, by performance measure, by utilization period and by managed element. The *c_Clandar* class and its properties assure their specification. Table 36 exemplifies a SLA, often used with our experiments, for a single base.

Table 36 : Base SLA specifications – Laboratory environment

Base size	Performance measure	Non utilization period (s)	Utilization period (s)
2 GB	Retrieval	5	1
	Calculation	50	5
	Restructuration	1000	100
	Data load	100	20

The utilization purposes (day/night cycle) are simulated by changing the date and time of the logical servers (the OS and the virtual server), such that we are not obliged to wait a full day. This way, with the theoretical experiments we reduce the day interval to up to 30 minutes (i.e. we can simulate 48 days/night cycles during a single day of theoretical testing).

Performance – the performance aspects relate to the definition of the user activity and SLA/Os. This aspect is common to the two environments, as the followed performance indicators are the same. The advantage of the laboratory environment is that it allows the isolation of specific indicators. This implies that we can perform the experiments for one single performance indicator at a time (i.e. only data retrieval queries are launched to obtain the query response time).

8.3.2 Real environments

Unlike the theoretical laboratory environments, a real environment describes what we face with our clients. A real environment is different in several ways, the most important being that it is unpredictable (prone to constant change). To point out the main differences, we retake each of the environments aspects.

The hardware architecture is characterized by the existence of several physical machines. The presence of virtual machines is a possibility also, but not so often met with our clients. The general component diagram of a client physical data warehouse environment is shown in Figure 86.

The physical servers are now separated; each of them has with its own (unshared) resources.

The logical architecture – each of the data warehouse architectures is specific to the physical machine on which the logical server is installed. There are logical servers that have only one physical application and logical servers with even as much as 30 applications. We have even met servers with more than 200 Essbase cubes. The characteristics of the logical architecture are:

- The version of the logical servers may vary on different machines (even though if it is not a usual case)
- The number of physical applications and their characteristics (size, number of bases etc.) vary for each logical server
- Each base is different, in data and size, as they are part of specific applications (e.g. Budget, Accounting etc.). Normally, a duplicated base will never be found in a real data warehouse architecture (unless specifically expressed by the requirements).

- The parameter configuration for each base, application or logical server are likely to differ and to reflect the level of usage and the expected performances (SLAs)

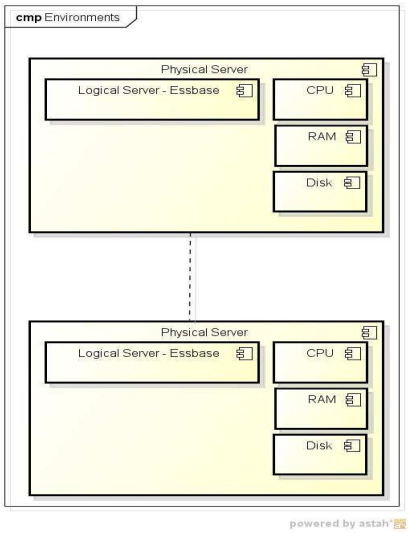


Figure 86 : Real environment hardware architecture

The user activity in real environments is generated by the real users, and proves to be quite unpredictable. As the DSS experts saying goes, give me what I want so I can tell you what I really want, the data retrieval operations in particular explode exponentially. We identify the three types of user activity specific to the: (i) normal user, (ii) system administrator and (iii) developer or DSS expert. Normal user queries usually revolve around the 1 s threshold, as also specified in the theoretical environment, where as administration queries can even last for tens of minutes (e.g. selecting all the cube data). Moreover, looking again over the four types of activity, most of the it (96,4 %) is generated by data retrieval queries. Figure 87 indicates the distribution of the user activity per type of activity over a one month period for a real Essbase logical server.

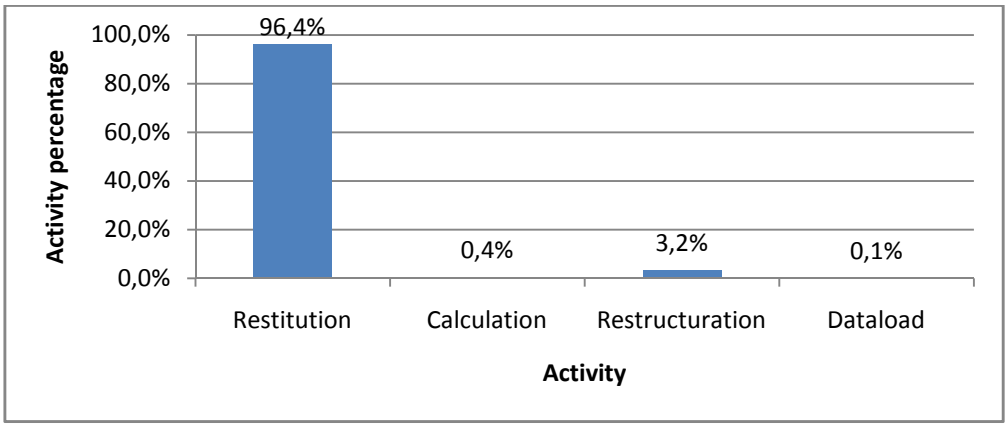


Figure 87 : Logical server activity distribution over a one month period

Restitution – that directly concerns the query response times. Similar to the theoretical environment, or the link that is the other way around, most of the user activity is done by

normal users requiring ‘small’ reports. The distribution of the response time query activity is shown in the diagram below for one logical server during a period of one week:

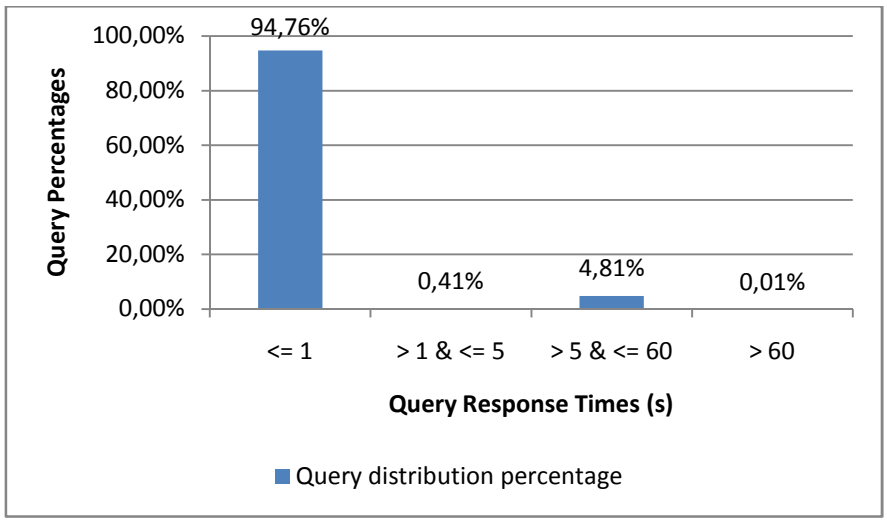


Figure 88 : Query response time distribution in a real environment

Calculation activity – Table 38 shows an example of the average calculation times in rapport with the cube sizes (for CALC_ALL operations – full cube recalculation). In comparison with the restitution times, the number of calculation queries is very small.

Table 37 : Average calculation time related with the base size for base for CALC_ALL

Cube size (GB)	Average calculation time (m)
15	33.3
2	14.6
0.149	0.8

Restructuration time – restructuration operations are second in the activity chart (3,2% of the total number of operations). They are influenced by several parameters such as: CPU power, RAM memory, base size, cache and block size value etc. We retake the table above and show the average restructuration times in correlation with the base size, along with the data load times (Table 38).

Data load – in real environments data load occurs rarely (only 0,1% of the operations) and most of times never during several months. Performance over data load operations is the less important.

Table 38 : Average restructuration and data load times with base size

Cube size (GB)	Average restructuration time (s)	Average data load time(s)
15	2289.3	79
2	383.7	18
0.149	4.7	0.9

SLA/Os – for the service level specifications we remain in the same area of utilization periods and purposes, as the elements don’t change form the theoretical environment. The

agreements are specified by the client, and they indicate the periods on which applications are mostly used. Usually they are elaborated by logical application. The big difference is that in real environments we don't have SLAs elaborated for individual bases. A typical client SLA example is: "I need the budget application to be optimal in the last two weeks of September; outside this period it must only be responsive". This translates into: find the optimal resource allocation when this application is in the utilization period, while in the non utilization period keep its configuration at the minimum required.

Performance – Performance considerations change for the real environments due to unpredictability. For instance, the days when developers have high activity on the cubes greatly affect the performances of the normal users. The main differences form the 'laboratory' environments are:

- Uncontrolled user activity – i.e. developer actions over the data warehouses, administration actions, users reporting explosion
- External factors – such as changes of the physical environment (change of the RAM, disk etc.) or change in the environment conditions (e.g. OS freezes, network or power failures etc.)
- Uncontrolled parameter reconfiguration – experts intervene on the data warehouse parameters, even several times per day if necessary. This can impact the performances greatly.
- Noise – the real environment performances present a higher noise level than the theoretical environments. The causes are multiple, from CPU fluctuations to network lags.

8.4 Conducted experiments and results

The conducted experiments and the results we have obtained with BI Self-X are divided according to the AC principles and the corresponding modules: self-: configuration, healing and optimization.

8.4.1 Self configuration

The self configuration experiments concern the pre-production configurations of data warehouses. Before a new data warehouse is put into production, a preliminary phase is required as to establish its starting configuration (parameters, allocated resources etc.). The BI Self-X Self Configuration module has the objective of finding a good (if not optimal) starting configuration, by using the best practices formalized in the knowledge base.

The theoretical and real environments are not separated with this experiment. They are similar because the concerned phase is pre-production (development), which allows a high flexibility. The methodology is to try several data warehouse configurations and choose the one with the best performance levels. The tested configurations include: the minimum/maximum resource configurations, random configurations, often met configurations and, of course, the BI Self-X configurations. With each of them, we performed the same series of tests to observe the performance.

The managed elements affected by the configurations are usually low level in the DSS hierarchy (most of the configuration is done at base level). The experiment we have conducted included the following:

- A physical server machine which was to host the future production data warehouses (8 GB of RAM, 2 TB hard disk, Intel quad core processors)
- An Essbase logical server (version 9), containing 10 physical applications and 38 bases.
- The interesting configuration parameters are(by managed element type)
    - Base
        - Storage Mode: aggregation or block
        - Caches: Index, Data and Data File
        - Access mode: Direct or Buffered
        - Block size value
    - Application:
        - Hard disk space
    - Logical server
        - RAM Memory allocated
- The Essbase readme documents and the expert's knowledge about the initial setup of the data warehouses (including best practices and recommendations).

The graph from Figure 89 shows how the average query response time changes with the different data warehouse cache allocations, under the same (simulated) activity conditions.

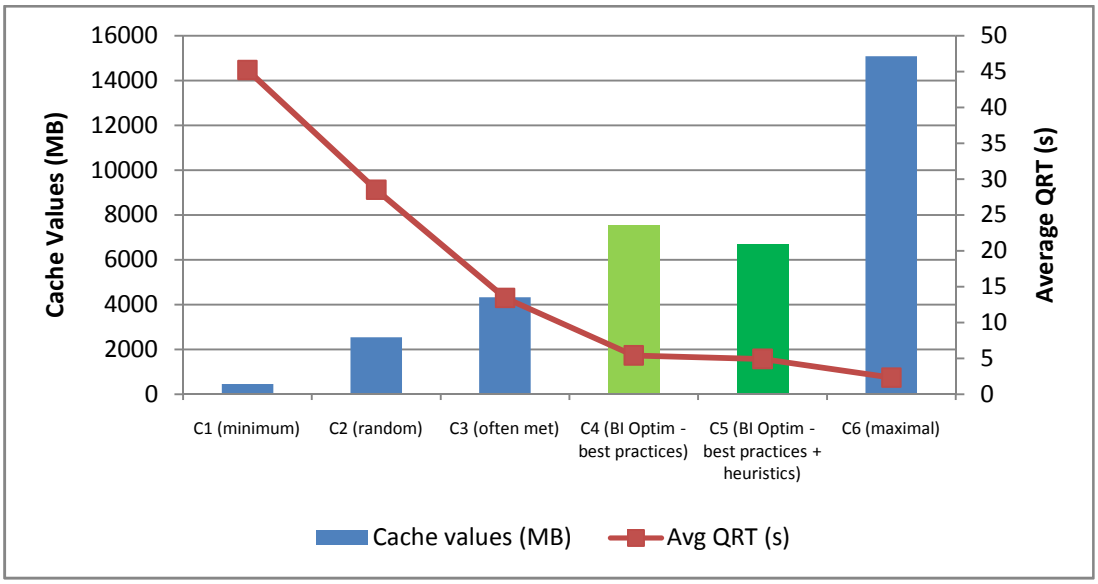


Figure 89 : Average QRT variation with different cache configurations

We have performed an activity simulation equivalent to a one day period and tested it over six different configurations. On the left vertical axis we have the allocated cache values and on the right axis the average query response time for the data retrieval operations. The six configurations correspond to:

- *C1 – the minimal* working cache configuration, following Oracle specifications (the Essbase v9 Manual). The minimal configuration also corresponds to the default cache configuration.
- *C2 – a random* configuration, generated in a random manner, between the minimum and the maximum configurations. Here the random has returned a low cache allocation. It was introduced in the experiment as a comparison point, but cannot prove to be a good reference due to its own state of 'randomness'.
- *C3 – the often* met configuration. This was established by our experts when asked to give a configuration for the data warehouse.
- *C4 – the best practices (business rules)* obtained by taking into account the best practices and pieces of advice. This is the first proposed configuration by the Self Configuration module. It takes into account the current environment and resource conditions, and, based on the knowledge base rules, it computes the corresponding parameter values.
- *C5 – the autonomic computing improved best practices* is the result of the second state of the Self Configuration module. The values are obtained starting from the C4 configuration, and trying to find a better combination (heuristically) around it. We label this configuration also as a result of the Self Optimization module, because it is obtained after a succession of several optimization attempts, specifically five.
- *C6 – the maximal* configuration, where caches are set at their maximum values. These values are either specified by the Essbase documentation (e.g. equal to the sizes of the bases) or limited by the available resources. We note that in our case the maximum configuration is not possible because there are only 8GB of RAM installed on the physical server (the performance was obtained on a 16GB server for comparison purposes).

As cache allocations increase, from 456 MB in C1 to 7524 MB in C4, the average response time decreases almost 9 times (from 45.2 to 5.4 seconds). We observe that between the often met configuration and the first BI Self-X Self Configuration proposition there is almost 150% response time gain (from 13.4 to 5.4 seconds). Even in its first state, with the best practice rules taken into consideration, BI Self-X offers *a 148% performance gain*. More interesting is the passage from C4 to C5. Even with a *12% decrease of the used memory* **(**from 7524 MB to 6688 MB), the average QRT has a variation smaller than 10% (of 0.5 seconds). C5 brings further the efficiency of the initial BI Self-X configuration, with self optimization, by providing a very good configuration / performance ratio (the best out of the 6). C6 obviously will provide the best response times (with the maximization of the cache allocations), but is not possible with the system, as the available RAM is limited to 8 GB.

The experiment has shown that starting configurations for new data warehouses can be substantially improved (up to 9 times if the default configurations are used). Many companies use the default configurations (C1) for their data warehouses and then they spend time and money on interventions from the DSS experts to reconfigure their systems. Others, demand this configuration in pre-production (C3), which is the case for most of our clients. With the usage of BI Self-X the C3 configuration becomes C4 or even C5 if optimization is applied.

Note. We remind that the self configuration experiment concerned the pre-production stage of the data warehouse. The objective was to find, if not the 'best', a very good starting configuration. Accordingly, the followed performance indicators were purely technical (without the consideration of the SLAs and the service levels).

8.4.2 Self healing

Self healing has two components: (i) the diagnostic phase, when anything that may be wrong with the DSS is identified, and (ii) the healing phase, where the diagnostics are solved by healing actions (the link between the diagnostic and heals). We consider that the hardest part is the identification of diagnostics, as healing is a matter of applying a series of specific actions in the specified order. Therefore, we present the results we have obtained with the self healing module in the light of self diagnostic.

As we have seen, the main objective of the self diagnostic is to reduce the time spent by a human expert doing the same job. This implies consequently the reduction of the financial cost and 'healing' time when problems occur and also a higher accuracy in the diagnostic. Our experiments focus in this exact direction. We first show the temporal gain obtained by using BI Self-X with self diagnostics, for then to greater argument this gain by introducing the financial aspect into the balance (in the real environments experiment).

8.4.2.1 Theoretical environment

The theoretical environment for the self diagnostic module allows to individually isolate each diagnostic and to validate each analysis rule. Whenever a new diagnostic or rule is introduced in the system, the theoretical environment permits its proper integration. The theoretical experiment was run under several stages in which the purpose was to see the differences in the time required by the human experts to pose the diagnostic and the time taken by BI Self-X. We must note that the expert we 'used' for our tests was also at the base of the ontology diagnostics rule elaboration, so all the existent diagnostics are supposed to be known to him, thus he should be able to identify them.

The set up physical environment contains:

- A single physical machine with virtualization support
- One virtual machine installed with a Windows 2003 Server OS, SRV_PHYS
- A single Essbase logical server, SRV_LOG
- Two psychical applications, APP_1 and APP_2
- APP_1 contains 6 data identical Essbase bases, BASE_11…16
- APP_2 contains 2 data identical Essbase bases, BASE_21,22
- Parameters for each base and applications are specific to the performed experiment and are presented accordingly

The first performed experiment was with a single diagnostic which referred to the base data block size dimensions. The rule that was followed was: *An optimal data block size should be between 2 and 100 KO, for BSO bases.* The rest of the DSS parameters were

configured in such matter that this was the only problematic diagnostic. Specifically we have configured two bases, BASE_11 and BASE_22 with a data block size of 500 KO.

Given the system, we asked our expert to identify the single problem with the data warehouses. He was given accessed to the DSS and was provided access to BI Copilot DBs over system configuration. In addition, he was specifically told that only one configuration problem exists. This was the simplest of possible tests, and the expert took 4 minutes to identify the problem and the two bases. In order to identify it the expert had to go through each configuration report for each base, application and the logical server. The BI Self-X self diagnostic module took 4 seconds to identify it and the time was due to the data load into the ontology from the BI Copilot DBs. *The temporal gain is substantial: 1 to 60, the BI Self-X system has given the information 60 times faster than the expert, on the simplest diagnostic experiment.* We also define the accuracy rate of the expert, based on the number of diagnostics identified in rapport with BI Self-X (in this case it is 1, as only one problem existed).

The second experiment was to test the efficiency over the entire set of diagnostic rules. We have configured the data warehouses in such a manner that the maximum of possible bad problems appear (or how a data warehouse configuration should NOT be). There are 19 total different diagnostics: 13 non-critical configuration problems (the data warehouses work but are not optimized) and 6 critical problems (the data warehouse don't work).

Given the environment, we asked the expert to identify the maximum number of possible problems. He was again given the BI Copilot rapport support, but was not specified the total number of existent diagnostics. After an analysis that took 10 minutes, the expert was able to identify 17 of the 19 diagnostics. The BI Self-X took 6 seconds. *There is a 100 to 1 gain in the analysis time, this time with an accuracy of 89,4%.* All though the time needed for the expert to identify the diagnostics decreased in rapport with the number of existent problems, the accuracy decreased as the expert was not able to identify the whole 19 diagnostics (which him self helped elaborate).

Figure 90 shows the results for the two tests, with the temporal gain and the accuracy over the two vertical axes, and the test conditions in the horizontal one.

Even if the accuracy dropped in the second case, the human expert identified successfully the 6 critical diagnostics, and, made a feedback with a new diagnostics which was not present in the knowledge base (and was afterwards integrated). This proves that the system doesn't replace the human expert, but improves and helps him with his work.

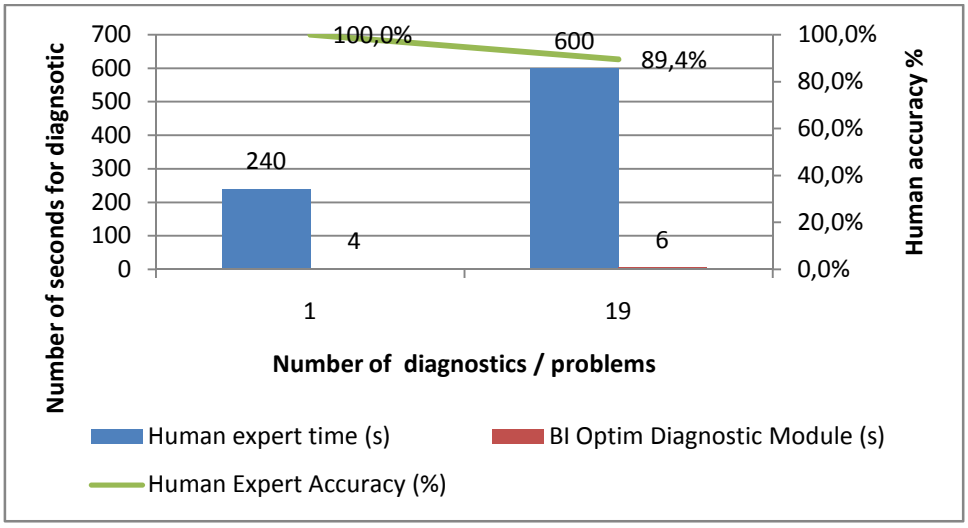


Figure 90 : Diagnostics elapsed time and human accuracy comparison

8.4.2.2 Real environment

The real environment is taken from our clients, and this time the objective was to see the *gain in time and money* of BI Self-X. In real situations, the client asks the intervention of the human expert if problems occur, therefore the human expert is in charge of fixing them. Healing implies that (i) a full diagnostic is performed or a diagnostic related to the problems indicated by the client and (ii) any possible cause of the problems must be fixed such that the problem doesn't occur anymore. The role of the BI Self-X self healing module is to help the expert with the first phase, identification of the causes of problems and of the possible healing actions. This implies a gain in the expert's time and consequently a decrease in the financial cost supported by the client.

The experiment was done with one of our clients, over the following environment:

- Three physical servers, each with a different power capacity (RAM, CPU and hard disk).
- The installed OS: one Windows Server 2000 and two Windows Server 2003
- Each physical server had one logical Essbase server. All the logical servers had the same major version (Essbase version 9)
- A total of 43 physical applications and 200 Essbases bases belonging to these applications, on all of the 3 logical servers
- Each individual base had its own parameters

Our human expert was called by the client indicating that there are extremely low response times when generating reports over the data warehouse ensemble. The client wasn't able to specify exactly which logical server was generating these problems and asked for a problem resolution.

From the point of view of the client expenses, the financial cost for the operation was established at today's market average: 1500 EUR / day, plus accommodation (if the case) fees and the time lost on getting to the location (indexed with the expert's hourly salary).

Our expert estimated that he required a full day to solve the problems. He had to drive 500km to the client s location, thus there was also a one night accommodation fee. In total, the client had to pay 2500 EUR for the intervention. On site, the expert spent 70% of his time to diagnose the problems of and 30% to fix them. Crossing this with the financial cost, *1750 EUR were spent for the system diagnostic*, whereas only 750 were spent for the 'healing'. If we consider nevertheless that the expert has to go on site for the healing operations (fixed cost), the *system diagnostic costs at least 1000 EUR* (70% of the 1500 EUR daily price).

From the perspective of the client, his access to the BI Self-X self diagnostic module gains him at least 1000 EUR per intervention. In average, from our experience, there are about 15 intervention needs per year for a regular client. For the client above, this means a yearly gain of at least 15000 EUR. This sum is somewhat low, if we don't take into account the *availability time gain*. Quantifying financially the availability time is a hard and subjective task. One possibility is to take the number of different users accessing the data warehouses in the non available interval (hours) and multiply it by the average hourly wage of these users. For example, for a 5 hours non availability, with 50 different users, with an average wage of 20 EUR / hour, the cost is 5000 EUR. In our example, we consider the70% of the 8 hours spent by the expert, thus 5.6 hours, 5600 EUR for each intervention day. Multiplied by the number of intervention days, the *availability time gain* is 84000 EUR. Adding up, there is a total hypothetical gain of almost *100.000 per year* for the client.

From the perspective of the expert, there are both positive and negative effects. The negative concerns the fact that BI Self-X will replace a good part of his work, thus will reduce his gains (instead of 1500 EUR he will only take 450 EUR for the 'healing'). Nevertheless, from the positive perspective the expert has an invaluable 100% accuracy tool that permits him the identification of the diagnostics in less than 1 minute. This gains him substantial time, allowing him to perform more operations in the same amount of time (i.e. over one month period). In addition, the commercial impact of an autonomic diagnostic module 'excites' the client and provides better business opportunities. Table 39 sums up the figures presented earlier:

Table 39 : Diagnostic expert time and client cost comparison for one year period

	Intervention cost (EUR)	**Non availability cost (EUR)**	**Total (EUR)**
Initial	37500	120000	157500
With BI Self-X Self Healing module	22500	36000	58500
Yearly potential gain	**15000**	**84000**	**99000**

This experiment shows that the two entities (human/machine) work together to faster solve the problems. The MAPE-K loop is assured by the software module (for the monitoring and analysis), and the planning and execution is assured by the expert himself. Evidently, some of the actins that are taken can be implemented directly in the loop, such that the expert only decides upon the modifications that will be performed, based on the 'healing' list proposed from the diagnostics. This reduces even further his time, and relates to the second aspect of self healing (action execution). Our future experiments will take this

into consideration too. Nevertheless, the objective is assured: the work time of the expert and the client costs are greatly reduced.

8.4.3 Self optimization

The self optimization experiments were conducted with the purpose of showing how the BI Self-X – Self Optimization module can improve the performance and service levels with existing production data warehouse implementations. The key word here is: 'production'. This means that the data warehouse architecture is deployed and data changes each day. Without a dynamic system that adjusts the parameters and resource allocations accordingly, the system degrades.

First, as we remember, DSSs function in a discontinuous way. The day/night discontinuity permits changing the parameters at the end of the day. The BI Self-X Self Optimization module changes each night the parameters of the data warehouses, accordingly to the implemented heuristics. Unlike the previous two modules, this time the MAPE-K loops are completely implemented by the managed elements ACMs, with the planning and the execution phases.

Second, as specified in the discussion about data warehouse performance, the most important are service levels, not the raw technical indicators. The objective of the BI Self-X Self Optimization module is to increase the quality of service offered to the user. Under the conditions of defining SLAs with regards to the utilization periods of data warehouses, we understand that we may have different QoS in different utilization periods, for different users and activities, and for different purposes.

For example, Figure 91 shows the difference between the raw performances of a data warehouse (specifically an Essbase application) and the QoS computed for three different criteria (and their combination). We assume the same data warehouse configuration is used over a 20 day period. This implies that the technical performance indicator (the query response time) is constantly the same throughout this period. However, during the 20 day interval, some days are more important than others, during which the data warehouse is considered critical. Therefore, the quality of service performance indicator and the levels of service perceived by the users are different during these days (even though technical performance is identical).

The purpose of the graph is to point out that users perceive differently an identical technical performance indicator, in rapport with their needs and expectations. For instance, having a 5 second QRT on an archive rarely used application is not that bad as having a 5 second QRT on a production constantly used application. This is translated with the usage of SLAs and SLOs, and by the specification of objective targeted values for each of the performance measure, for each of the criteria. By computing the QoS as the rapport between the objective value and the actual value, different values are obtained indicating the different levels of service corresponding to the specified criteria (e.g. the light blue QoS curve).

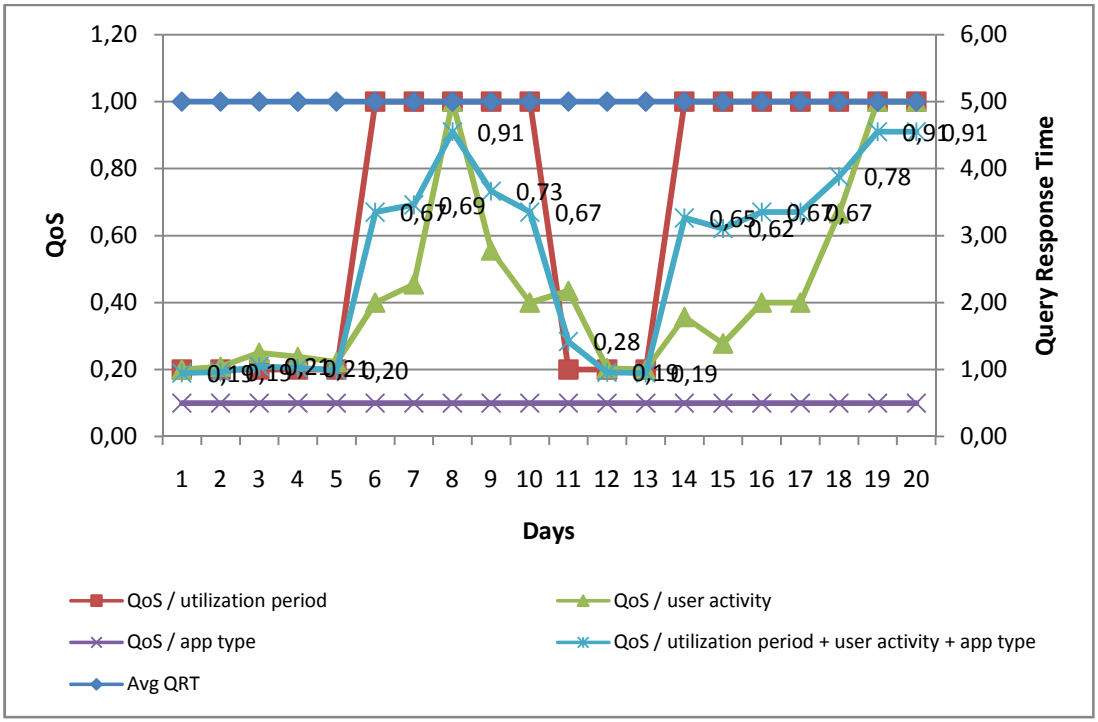


Figure 91 : Performance / QoS difference over a 20 day period

The right vertical axis contains the raw performance indicator, QRT, which, as reminded, is constant. The left vertical axis presents a combination of the quality of service depending on three chosen criteria: (i) utilization period, (ii) user activity and (iii) application type (and the three combined). Any QoS equal or greater than 1 implies a total user satisfaction. On the horizontal axis we have the 20 day period. Detailing over the three QoS computation criteria, Table 40 exemplifies the QRT objectives for each of them:

Table 40 : Target QRT with QoS criteria example

Criteria	QRT Objective (s)	QRT Value (s)	QoS = Objective / Current value
Utilization period		5	
High	1s		0.2
Low	5s		1
Application type			
Production	0.5		0.1
Archive	10		2
User activity			
100 queries	5s		1
1000 queries	1s		0.2

- *The utilization periods* – high utilization periods are specified between days 6-10 and 14-20. During these periods, referring to the target QRT table, the objective is 1 s. Similarly during low utilization periods the objective is 5s. Consequently, users perceive the 5s QRT value as unsatisfying during the high utilization periods (QoS = 0.2), while being completely satisfied during the low ones (QoS = 1).
- *The application type* describes how important is an application to the user, based on its purpose. For instance heavily used production applications are more important

than the archive applications. The type of the application usually doesn't change with time, therefore its importance from this point of view is always the same. In the presented graph, the data warehouse corresponds to a production application, therefore with very high expectations (0.5s). Therefore, the 5s QRT value is very unsatisfying to the users, reflected by a very low QoS (0.1).

- *The user activity* influences the perceived user performance proportionally with the 'size' of the activity. An application with a lot of activity (i.e. a high number of retrieval queries) is more important than an application with low or unexistant activity. The interval unit of activity is established to 1 day, and based on the number of queries executed during the day users perceive QRT as more or less critical. Objective QRTs scale (non linear), for example for 1000 queries being 1s. In our graph, the activity peaks correspond to the utilization periods (the highest during days 8-10 and 14-17). During the 20, we have considered a constant variation of the activity for each day, thus the activity curve changes a lot.
- *The combined QoS* is the final user satisfaction indicator, computed as a scaled average of the three QoS indicators presented earlier. As we have seen from the performance measure section, each of the composing QoS plays a role in the final user satisfaction, translated by a scale factor between 0 and 1. In our case, the utilization periods contributed with a scale factor of 0.5, the user activity 0.4 and the application type 0.1. This indicates that utilization periods play the most important role with the user satisfaction. The activity of the user has a smaller impact, but still important, while the application has the smallest influence. Combining the three curves, we obtain the combined QoS curve. It reinforces the initial premise: that QoS may vary strongly under the same configuration/technical performance conditions.

There are two objectives for the BI Self-X Self Optimization module. The first is one is to allow data warehouse systems to change their configuration parameters while in production. As we remember, the big problem with data warehouse systems is that they don't adapt accordingly with the changes in their size, data and usage conditions. BI Self-X reconfigures them in an automated manner by *taking into account the SLAs and SLOs*, a condition without which managing a DSS efficiently is not possible.

The second objective is to argument the fact that improving technical raw performances only can be a vital mistake, for its impact with the service levels is not direct.

Note. In any of the two objectives, we have to keep in mind that what we propose, both technical and service oriented, is a novelty with DSS management. To this point, there is no equivalent of the self optimization module, thus there is no well documented optimization (only small 'hacking' scripts developed locally by DSS experts). Monitoring DSS activity and integration of the SLAs are some of the novelty element introduced by SP2 and used by this thesis. Therefore, the self optimization results contain three dimensions: the situation of yesterday (and of today for many of our hopefully future clients) without the solutions of SP2, the situation of today with the BI Copilot support, and the situation of tomorrow, based on BI Copilot with BI Self-X.

8.4.3.1 Theoretical environment

The theoretical environment, in the case of the Self Optimization module, provides a great advantage over the production environment: potentially fatal changes don't affect the production process. The self configuration and self healing module concerned either the pre-production process or the production process with controlled changes (by the DSS expert). The self optimization module first performs the changes and then looks what happens during the next cycle. This can raise dangerous situations where a bad configuration can drastically affect the performances and even render the data warehouses useless. It is more or less like saying: "Shoot first, and then ask the questions". The theoretical environment allows us to refine the heuristics and to adjust the critical thresholds.

We present two experiments, aimed at the two specified objectives of the self optimization module. The theoretical environment is similar to the previous one used in the other modules. It is described by:

- A physical server equivalent – which is a virtual machine such that we control the CPU, RAM and disk capacities. This would allow us to play on the three important external parameters. These there are modified manually, thus are not part of the MAPE-K loops. Among the three, the most interesting is the RAM capacity, for which we have allocated 8 GB.
- One Essbase logical server (v9), which has 7.5 GB of allocated RAM memory, as 500 MB were reserved for the OS (a Windows 2003 Server SP2)
- Two applications, one with two Essbase cubes and the other with six cubes. The bases are identical in data and size, but differ in how they are configured. The size of the bases is 2000 MB each, with an index size of 300MB and a data file size of 1700 MB.

The theoretical environment enables us to specify the SLA/SLOs per base. Taking into account the four different types of activity transposed to performance indicators, an example of SLOs for a single base that we used for the experiment is shown in Table 41

Table 41 : Base SLO example with performance over utilization periods

Performance indicator (time)	SLO in utilization period (s)	SLO outside utilization period (s)
Response	1s	5s
Calculation	5s	25s
Restructuration	100s	1000s
Data load	20	200s

This kind of table is specified for each the 8 bases. Alternatively, the table can be specified at a superior level (e.g. application) and all the elements underneath inherit the SLOs.

As this is a theoretical environment, we retake the activity simulator to perform the four activity operations. As we remember, there is a percentage distribution for the four operation types, specifically: retrieval – 80%, calculation – 6.67%, restructuration – 10%,

data load – 3.33%. This helps us compute an overall technical performance indicator as a scaled average of the four, expressed as:

$$OverallPerf_{technical} = \frac{0.8 * Ret + 0.067 * Calc + 0.1 * Restr + 0.033 * Datal}{4}$$

Consequently, the overall QoS is computed in the same manner. The scaled factors remain the same, whereas the four individual QoS values are computed based on the SLA utilization periods. Several parameters influence directly the performances; the parameters that we used in the experiment are divided between manually modified and autonomic modified. The manually modified parameters are the 'external' linked to the characteristics of the physical server, mainly:

- *The available RAM* – which we can change only manually by modifying the virtual machine RAM allocation
- *The allocated RAM* – the amount of memory (from the available RAM) allocated to the logical server (Essbase).
- *Number of allocated CPUs* – for the physical server and for the logical server (e.g. the physical server may dispose of 4 CPUs, but Essbase uses only 2).

The autonomic parameters are changed directly with the MAPE-K ACM loops. They are 'internal' parameters, for the logical server and they relate to:

- *Storage mode* – specific to an application (thus to all the bases contained by the application), either block storage mode (BSO) or aggregation storage mode (ASO). The choice of the storage mode influences some of the other parameters.
- *Cache values* –for BSO, the three values of the Index, Data and Data File cache, for each Essbase base. For the ASO mode, the value of the ASO Cache, specified also per base.
- *Block Size* – the size of each data block, specific to each Essbase cube for the BSO mode only.
- *Fragmentation Ratio* - the level of disk fragmentation, specific for each base, which can be controlled by launching periodical automatic defragmentation.
- *Access mode* – specific to an application (and all the bases contained); it can be either Buffered or Direct I/O. The buffered access mode uses a memory buffer (thus consumes more memory) with increased performances. The direct I/O accesses directly the hard disk, thus relies on a good fragmentation of data.

8.4.3.1.1 First conducted experiment

The first conducted experiment, with the environment above, shows how the BI Self-X Self Optimziation module offers a constant adaptation of the data warehouse configuration parameters. We don't take into account the SLAs, and only aim at improving the raw technical performances.

Figure 92 shows how optimization of caches influences the average QRT for two data warehouses over a 15 day period. The objective of the graph is to show that by recurrent operations, the system reaches a balance state of the configuration and performance.

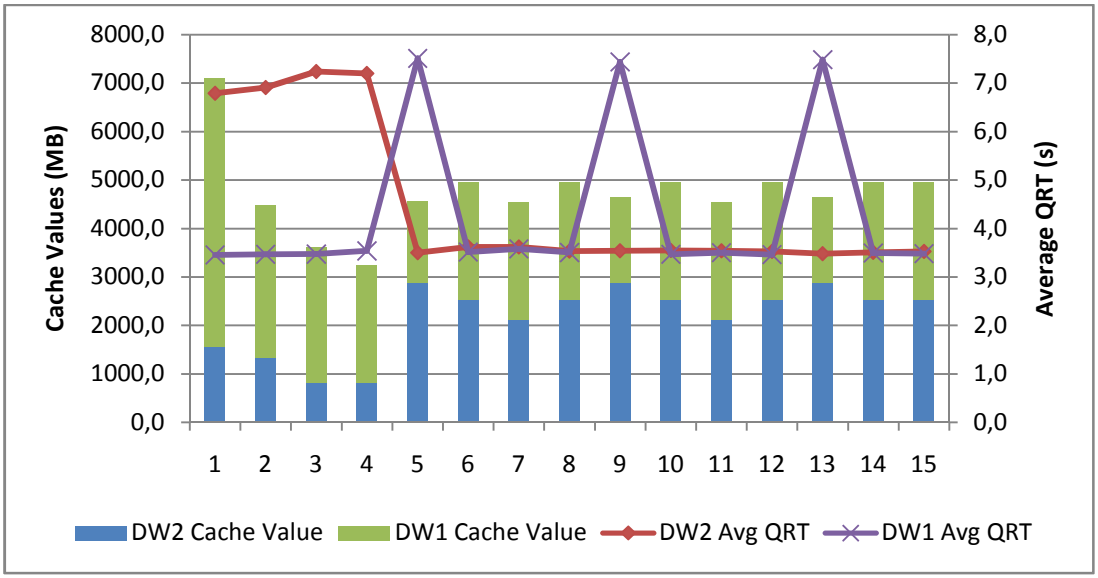


Figure 92 : Data warehouse technical performance improvement with BI Self-X Self Optimization

Without the BI Self-X module, the configurations would have not changed. The two data warehouses would have occupied 7000 MB of RAM and performances would have degraded as new data is added periodically. The configuration/performance ratio from day 0 would have been practically the same for the 14 days (with an eventual degradation in performance). On the other hand, with the adoption of the self optimization module and the heuristics for self improvement and resource reallocation, the graph shows that better resource / performance ratios can be obtained.

Referring back to the self management heuristics, in this particular experiment, the individual heuristic ran each day, while the reallocation triggered once each 5 days. This means that data warehouses decrease each day the cache/QRT ratio by gradually decreasing the values of their caches, while keeping a certain level of performance. Moreover, once each 5 days a reallocation of the freed memory is done between the two data warehouses depending on the individuals QRTs and their deviation form the average QRT. As we only have 2 data warehouses, at each reallocation one will have a high performance level and the other a low one.

In the first 4 days, the occupied cache decreases for both of the data warehouses, as only the individual heuristic is running. QRTs with DW1 are only slightly affected by the cache decrease from 5550 to 2429.1 MB (there is an increase of about 10%). This implies that DW1 had too much unnecessary memory for its caches. The same consideration is valid for DW2, as reducing its caches by almost half, the QRT only slightly increases. So, the evolution from the first four days indicates that both data warehouses had cache memory to spare.

Day 5 is the first day in which reallocations are made, therefore the freed memory from the first four days is allocated to the needing data warehouse (here DW2). Once the reallocation is made, there is an important performance boost for DW2 (almost 90%), where as DW1 has a sudden drop of performance due to the too low cache allocation (the peak

corresponding to 1681.7 MB). The performance boost DW2 indicates the fact that a high infusion of cache memory was needed in order to descend to the performance levels of DW1. Moreover, as the drop of caches from DW1 was too significant, DW1 switches places (from the performance point of view) with DW2, by showing a doubled QRT (which is unacceptable).

Starting from day 5, the self optimization continues to revolve around the same point, with attempts to further improve DW1 and DW2. This leads to some recurrent peaks when cache changes are not acceptable. Day 7 offers the best parameter/performance ratio, which translates into *a gain of 70% of the used resources and almost 100% for the performance*. Referring to the recurrent peaks, the way the heuristics are described doesn't implement an attenuation mechanism. Once a balance point has been reached, the heuristics will further attempt improvement. In the case above, each time the self improvement heuristic tries to decrease the cache value for DW1 right before a cache reallocation, performance drops greatly and the configuration is restored to the previous one. The self improvement heuristic enters an infinite loop as long as the utilization activity and data doesn't change. Nevertheless, this scenario is very unlikely with real applications, as user activity and data change all the time, therefore the same environmental conditions will most likely never happen.

Comparison point

In order to show how the BI Self-X Self Configuration module positions itself we present in Figure 93 a comparison between (i) the results presented above with the adoption of the BI Self-X module, (ii) random configurations and (iii) the state without the usage of the BI Self-X module or any other module. As the comparison between the with and without BI Self-X is demanded, we introduced the random comparison point to show that acting upon the configurations without knowledge of the system can prove 'devastating' for performance.

The comparison point is a ratio between the cache and the average response time, computed as:

$$Ratio = \frac{1}{Total\ Cache\ Value\ *\ Avg\ QRT} * Scale\ Factor$$

As the optimization aims at lowering the cache and the average QRT values, a higher ratio indicates a better situation. Consequently, we notice on the BI Self-X curve that the maximum ratio values correspond to the 'best' configuration day shown earlier: day 7. Moreover, the days during which we had the performance drops correspond to the ratio drops (days 5, 9, 13).

Comparing the BI Self-X curve with the situation where no action is taken for optimization, there is a clear improvement. The grey curve decreases slightly but constantly, as more data is added to the data warehouses. As the cache configuration doesn't change, performances degrade slowly. Compared to the results of BI Self-X, the ratio is improved more than twice. The random curve offers both better results (day 5) and very low ratios

(days 9, 13 etc.). It proves that it is not a viable alternative, even compared to the grey curve.

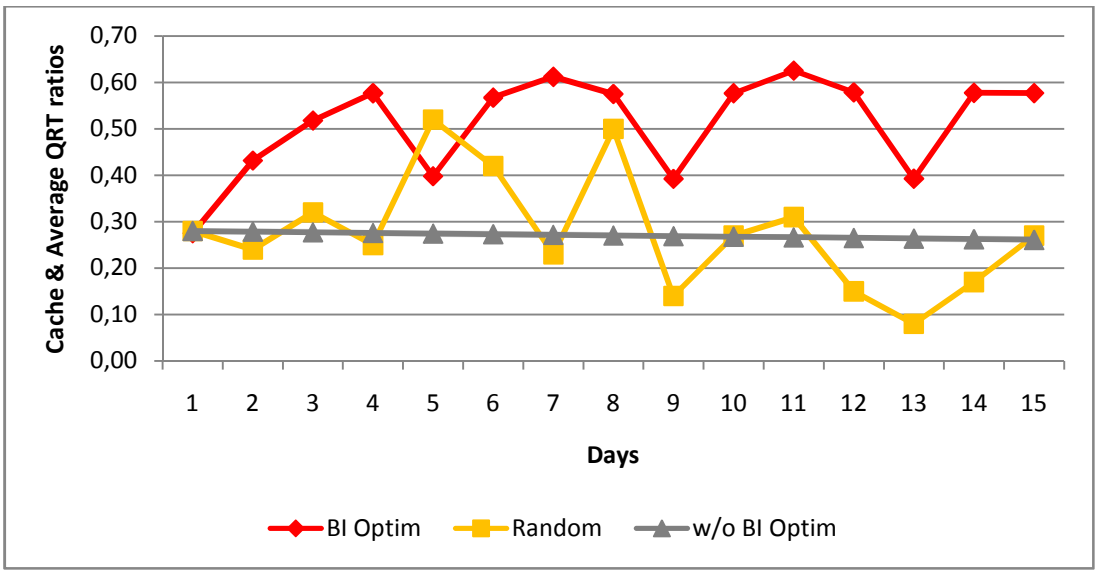


Figure 93 :Self-Optimization cache/QRT ratio comparison for theoretical environments

The objective of this first experiment is reached. We have shown that better technical performance and configurations can be obtained heuristically and that data warehouse management system should implement a constant monitoring / improving feature.

8.4.3.1.2 Second conducted experiment

The second conducted experiment has the objective to show that the key to an efficient DSS is not the improvement of the technical performance but of the levels of service. We wanted to show and to reinforce the fact that *not taking into account SLAs when trying to optimize data warehouses can prove to be the principle mistake with DSS management.* We retake the conditions from the first experiment, but this time we take into account the SLA activity periods and the target performance objectives for these periods. This way, the heuristics will not search to improve the technical performance QRT but the QoS indicator computed with regards to the specified SLAs The results are shown in Figure 94.

On the left vertical axis we have the total allocated cache, whereas on the right vertical axis we have in red the average QRT and in green the average QoS. The QoS indicator us scaled such that it fits the same metrics as the QRT. We have done this shift of the QoS curve to better show the ‘intersection’ between the QoS and the performance and to underline the fact that better performance doesn’t necessarily imply higher QoS. On the horizontal axis we have a period of 30 simulated days, with the high utilization periods between days 5-10 and 12-15. The simulation is the same used for the first experiment.

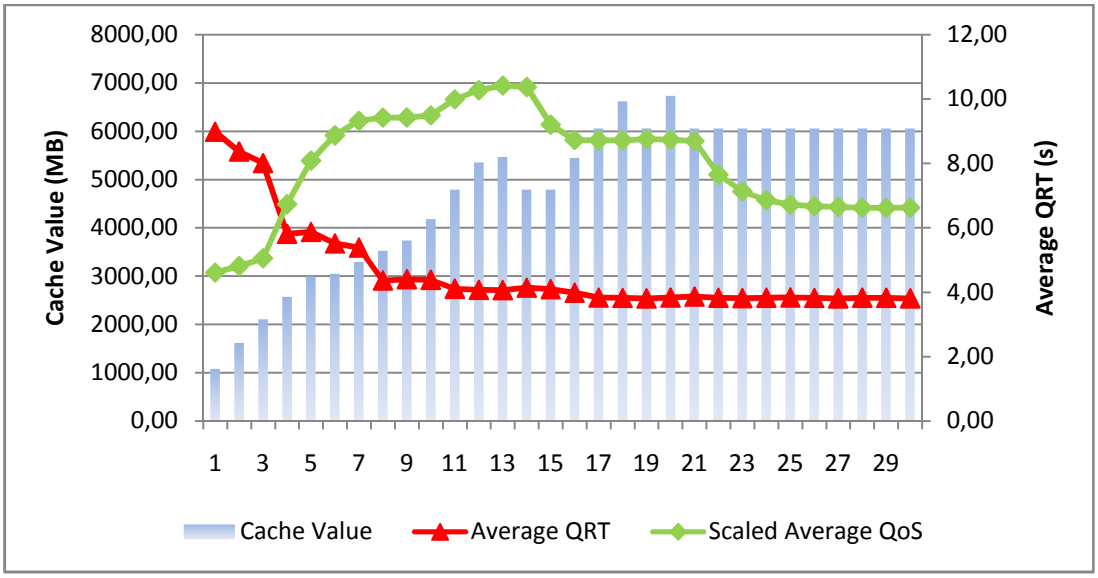


Figure 94 : Query response time and QoS evolution with cache allocations

This time, our experiment started with an initial minimum cache configuration, where we have a high average QRT and a small QoS. Each day we try to improve the cache/QoS ratio (with the individual self improvement heuristic) and each 5 days the reallocation heuristic is triggered. This occurs on the days corresponding to the peaks of the QoS curve (5,10,15,20). Towards the end of the month the system gradually stabilizes with the cache allocation, QoS and QRT. The 'best' found configuration is during day 13, the criteria being the maximization of the QoS. The experiment reinforces again the fact that good technical performance doesn't necessarily mean a high QoS. *The highest QoS* is obtained for day 13 with a corresponding *QRT that is higher than the minimum obtained QRT*. This simple fact shows our objective for data warehouse management: increase of the levels of service.

Comparison point

Similar to the first experiment, we provide the comparison point between (i) the BI Self-X Self Optimization module configurations from Figure 94, (ii) random configurations and (iii) the configurations without any optimization module. The three curves are shown in Figure 95.

This time, the comparison point is the average QRT and its evolution over the 20 days. The lower it is the better performances are. By showing the differences in QRT we understand implicitly the differences in the QoS for the three cases.

In the case where no optimization module is used (the grey curve) we see that performances degrade slowly but constantly, due to the increase in size of the data warehouses. Compared to the BI Self-X curve (red), we have QRTs that are more than two times smaller, starting from day 10 and with the performance gap widening. Relating to the random curve, most of the QRTs are higher then both the BI Self-X and the w/o BI Self-X curves. Similarly, the conclusion is that modifying parameters without knowledge of the system leads lower performance.

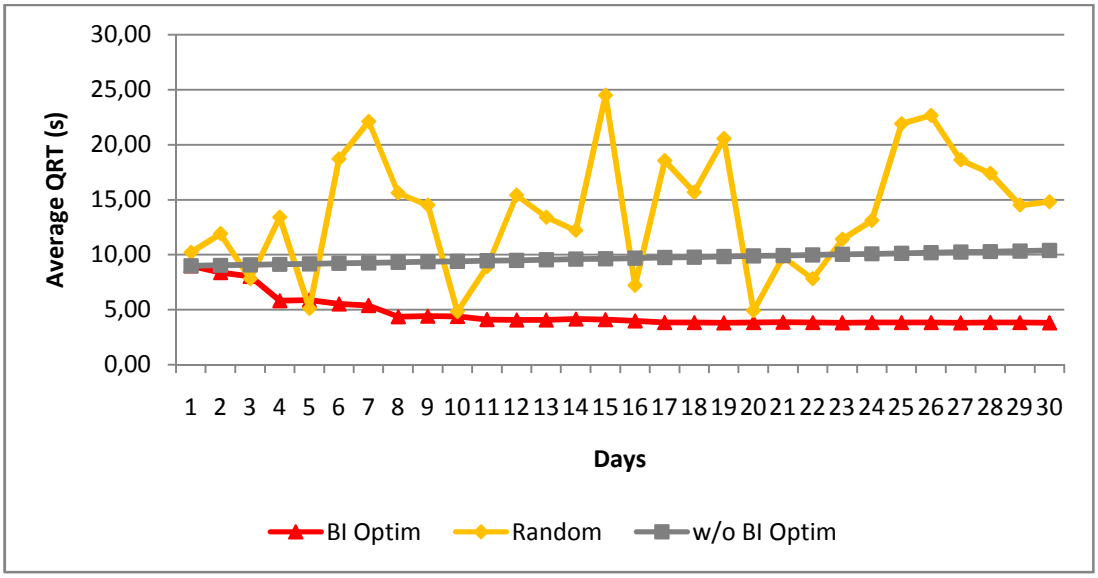


Figure 95 : Self-Optimization QRT comparison with SLA considerations

8.4.3.2 Real environment

The theoretical experiments had the objective of assuring the two specified objectives for the self optimization: the benefits of integrating a continuous monitoring and optimization system with data warehouse and that performance with data warehouses must be indicated by the quality of service, depending on the SLA/Os, and not by the raw technical indicators. The results were shown on a restraint perimeter with a few data warehouses, configuration parameters and performance indicators.

For the real experiments, we have tested the module on a client environment. The big constraint was regarding the critical changes, as we have shown that the self optimization works on the principle: act first and then see the results. With the theoretical environment critical points were not a concern, but with the real production environments we cannot afford any 'diversions'. Therefore, the heuristics modification ratios and thresholds are greatly reduced such that the change impacts of the parameter modifications have smaller impacts. This way, even if the QoS risks of heavily dropping during certain days, it will not reach 0. A QoS value of 0 implies that the system is no longer working.

The real environment that was used for testing contains:

- A real physical server dedicated to the storage of the data warehouses. It disposes of 16GB of RAM, from which 500 MB are allocated for the OS (Windows 2003 Server SP2).
- One Essbase logical server (v9), which uses the rest of the 15.5 GB of RAM.
- A big production application, containing 12 different bases, with a total files size of 100GB.

With the above configuration, we have run the BI Self-X Self Optimization module for a period of 20 days. During this period the activity was assured by the users of the

application, knowing that the utilization period doesn't change. Although, this may seem contrary to the statement of SLA use, even without the changing of the utilization period there we have obtained notable improvement in resource usage, technical performance and quality of service. The following parameters are concerned with the experiment:

- *Configuration*
  - Total cache values: the sum of the three caches (index, data and data file) used by the data warehouse
  - Block size: the block size for each base of the application. In this case it was common for all the bases of the application
  - The access mode: again common for all the bases of the application. It is configurable only at application level.
- *Performance*
  - The retrieve time (QRT) for building rapports with the base data
  - The calculation time for the reallocation of the base dimensions.
  - The quality of service (QoS), as a combination of the performance indicators and the used system resources (the RAM used)

From the heuristics point of view, the behavior is similar to the theoretical environment. The individual self optimization heuristic runs daily, while the self configuration reallocation heuristic applies once each five days. The two heuristics apply for the cache configuration parameter, over all three performance indicators. As reminded, the step and threshold are significantly reduced in comparison with the theoretical environment, to prevent big performance drops. For this experiment we have used a step value of 0.04 with an acceptance performance drop of 0.02. Correspondingly, we have divided in half the values from the theoretical environment.

We present the obtained results with the help of several graphs, organized by parameter and followed objective, while reminding that the primary objective is the improvement of the QoS.

First, Figure 96 shows how performances (calculation and retrieval) are improved while decreasing the cache usages. Consequently, the QoS increases as the two durations and cache usages decrease.

We notice that cache values don't change too much as restricted by the heuristics low thresholds. The average QRT follows a somewhat constant pattern, with a slight improvement in rapport with the initial value. Between days 14 and 18 the cyclic behavior seen in the theoretical environments occurs, as the balance point has been reached. The calculation time curve is similar to the QRT curve, but with a better overall gain from the cache modifications (from day 5 on). The cyclic behavior is present equally, with the mention that calculation impacts are more powerful than retrieval impacts.

Nevertheless, the objective is achieved, as better response and calculation times are obtained with fewer cache memory, thus the QoS is improved.

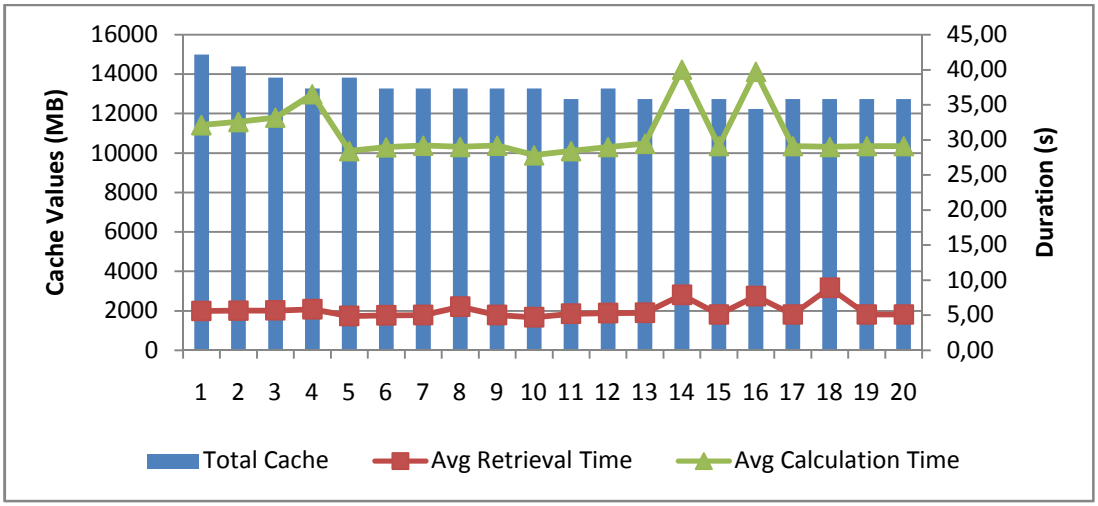


Figure 96 : Self Optimization performance with caches on a real environment

Second, we choose another parameter, the block size, to control its influence on the performance durations (Figure 97).

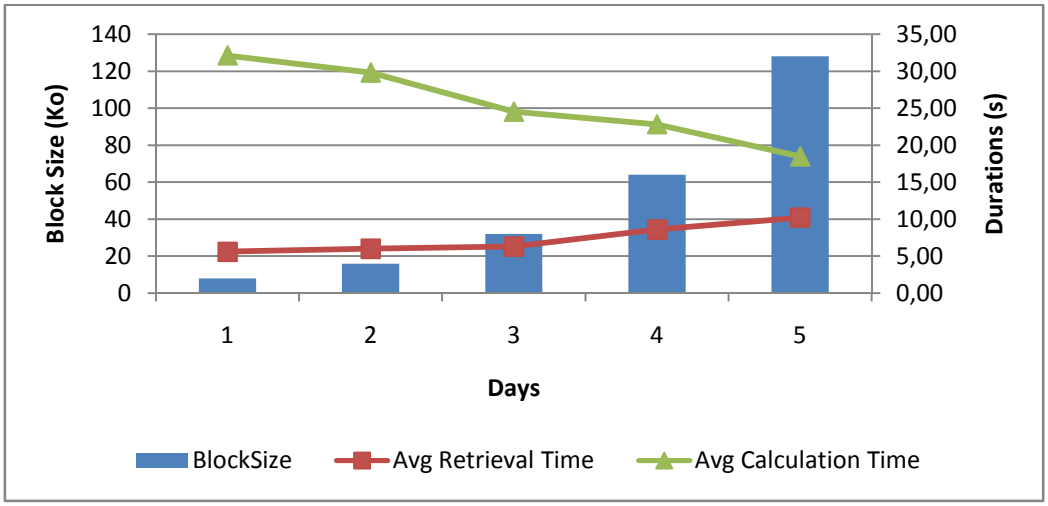


Figure 97 : Self Optimization performance with block sizes on a real environment

There are five different modifications that are made for the block size, with inverse impact on the calculation and retrieval time. As the block size doubles each day, the retrieval time increases whereas the calculation time decreases. Thus the impact on the overall QoS is remains somewhat constant. Depending on the followed purpose, this could change. For example, if the usage period is night (where only calculations are made), than there is a big increase over the QoS.

Third, and last, we analyze the impact that the access mode has on the performance levels in Figure 98.

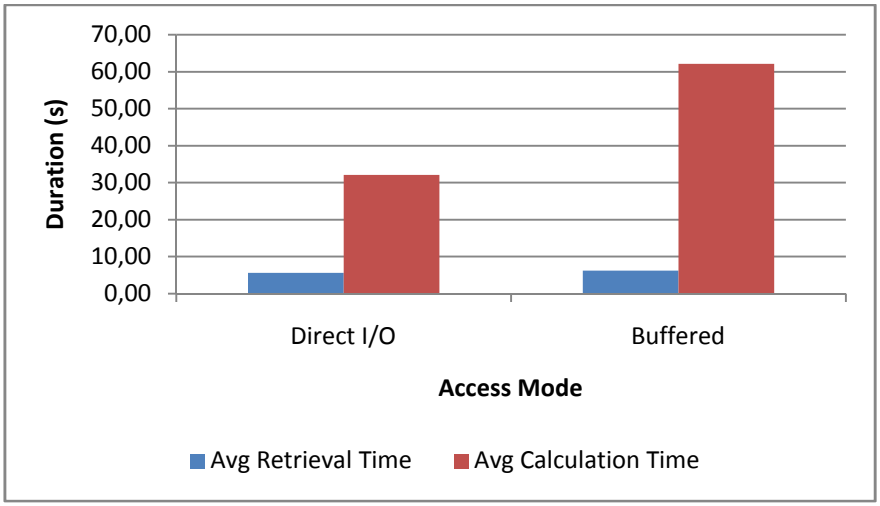


Figure 98 : Self Optimization performance with access mode on a real environment

In this specific case, as the disks were very fast (15.000 RPM), the impact in the retrieval time is basically negligible. Nevertheless, the calculation time is strongly impacted by the buffered mode due to the large quantity of data.

Comparison point

Once again, we show in Figure 99 the comparison between the results obtained with (i) the BI Self-X module, (ii) random configurations and (iii) without the usage of the BI Self-X module. We note that this time the differences between the with and without BI Self-X curves are much more smaller, as it is a production environment and change intervals decrease drastically.

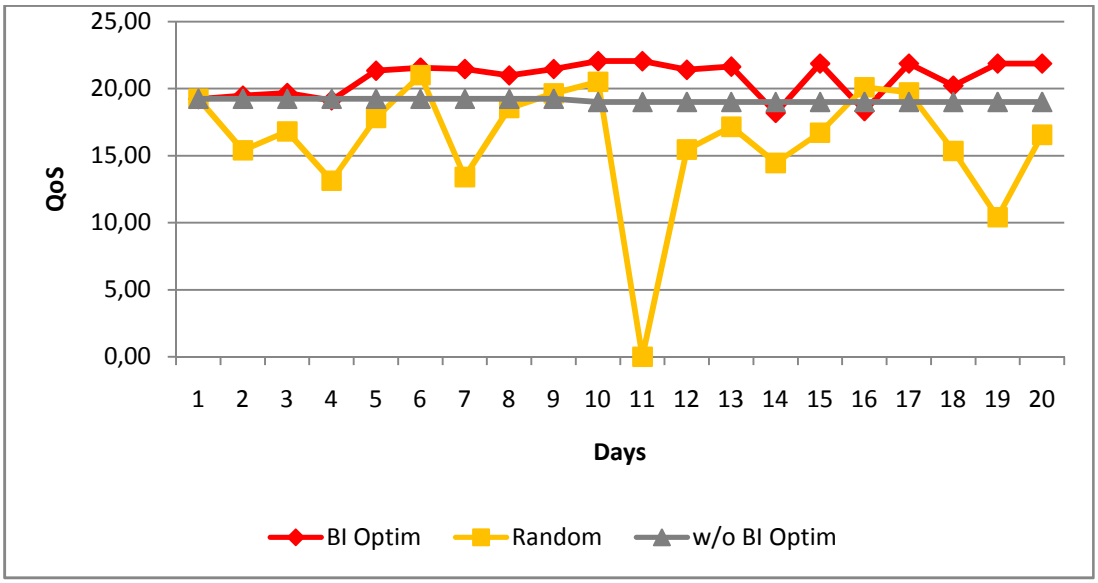


Figure 99 : Self Optimization QoS comparison for a real environment

We considered as comparison point the Quality of Service final indicator. The QoS was computed starting from the results presented earlier, by combining the total used cache, the

average retrieve time and the average calculation time. The used formula is described below, where *SF* stands from '*Scale Factor*'.

$$QoS = \frac{1}{CacheValue * 0.5 * CacheSF + AvgQRT * 0.3 + AvgCalc * 0.2} * GlobalSF$$

Comparing the red and grey curve, the QoS gain for the BI Self-X module is smaller reported to the performance gain in the previous experiments. Moreover, during certain days we even obtain slightly smaller QoS values (days 14 and 16). As mentioned, in real environments permitted changes are much more restricted to prevent system failures. Such a failure was obtained in one of the random configurations : day 11. Because of a bad access mode / cache value combination, the calculation operations didn't end, such that the calculation time was considered infinite. Consequently, the QoS was equal to 0.

Concluding the real experiment case, the objective is reached by providing better configuration/performance ratios for production environments. Even if the gains are smaller than in theoretical environments, we have shown that by using the BI Self-X Self Optimization module, enterprise scan further increase the quality of the provided service of their data warehouses.

8.5 BI Self-X approach interest

Summing up the experiments and results presented earlier, we discuss in this sub section the impact he BI Self-X has both with the solutions offered by SP2 and in the context of the client.

SP2 Solutions interest

The initial BI- Copilot offered solution was designed as a surveillance and reporting tool. It basically performed the first step of the autonomic computing model: monitoring, offering no 'in house' analysis (unless explicitly audit operations demanded by the client from our experts). Moreover the information and data gathered by BI Copilot referred to the organization, configuration and performance of the DSS and the data warehouses only. No additional meta-information was contained (such as best practices, business rules etc.).

BI Self-X brings two main new elements to BI Copilot. *First*, it a complete integration model for all the knowledge involved in managing a DSS, from configuration and performance parameters (already present) to best practices and policies implementations, SLA adoption and QoS assessment (BI Self-X). The ontology semantic model allows the integration and formalization of a much wider spectrum of information sources, from readme technical documents to the experience of the DSS experts themselves.

Second, BI Self-X closes provides the self-management capabilities that impact the DSS directly. By using a an entire series of rules over the data monitored by BI Copilot, it is capable of autonomic diagnostics and even heal actions. It basically closes the MAPE-K loop offering a complete self-management solution. This, as we have seen from the experiments, leverages a good part of the expert's work, reduces his intervention times and helps him to be more efficient.

Market / Client interest

From the market and client point of view, BI Copilot was a very innovative product, in the sense where nobody before SP2 was performing aggregated monitoring tasks over DSS environments.

Therefore, from the client point of view, BI Copilot lifts his/her head from the ground, whereas BI Self-X helps him look up to see the sky above the trees. Nevertheless, from the return over investment results, we have seen that using the BI Self-X Self Diagnostics module greatly increases the availability time with data warehouses, thus reducing client loss and expenses. Metaphorically, BI Copilot says: '*You are lost, I can show you the way'*, whereas BY Optim adds: '*Now that you know the way, I can fly you there*'.

From the IT management market perspective, BI Self-X intrudes a new term as an evolution of ITIL's CMDB, that is the: *Configuration Management Knowledge Base (CMKB).* The CMKB indicates that management takes into consideration all the information available, both data and meta-data and in addition offers semantic and collaborative capabilities. Developing over this concept is part of our future works, and we are convinced that is the next fundamental step of system management. Therefore, this provides an excellent continuity point for future works on the subject of DSS management.

8.6 Conclusion

This chapter presented the experiments and the results we have obtained with our approach. The first part provided a detailed description over the BI Self-X software prototype, including its three modules for configuration, healing and optimization. We have shown how it can integrate with existing data warehouse systems, and how the different components interconnect.

The second part focused on the description of the experimental environments (theoretical and real), their specifics, and how user activity is distributed and simulated. For the data warehouse implementation we have used the Oracle Hyperion Essbase BI solution.

Finally, in the third part we have shown a series of experiments and results that we have obtained for each of the three modules. The experiments have been made in each case over a theoretical environment first, to test the effectiveness, and then over a real 'client' environment to show the prototype capability and behavior with higher charges and unpredictable environments.

From the comparison point of view, we can assess that performing the experiments was fairly in our advantage, as so far up to this point there are was existent integrated solutions for DSS management. Everything was performed manually, from monitor to analyze and interventions over the data warehouses. A comparison is not into question, but being pioneers in the domain offers this advantage (and disadvantage as we step on unexplored ground).

Overall, the results have shown that it is always better to use a machine or software (BI Copilot) where possible over the human factor which is expensive, much slower and

potentially unreliable (which is fairly obvious). We have shown with our solutions from this thesis that there is place for improvement, (BI Self-X over the BI Copilot suite). Moreover, the results reinforce the advancement towards both data warehouse client satisfaction and DSS progress.

9 Conclusions and Perspectives

"In my end is my begging"

T.S. Elliot

9.1 Conclusions

This thesis combined three major research areas: (i) management of Information Systems, specifically *Decision Support System and Data Warehouses*, (ii) knowledge formalization for managing a DSS via web *Semantic Web* Technologies and Ontologies and (iii) autonomic task management with *Autonomic Computing*. These are the three pylons of our work.

Decision Support Systems and Data Warehouses offered a unique and novel applicative case. The state of the art allowed us to understand the situation today, what the main problematic points are and how our approach can help solving some of them. From the principle that you cannot manage what you don't measure, DSSs lack a proper assessment of their environments. A parallel was constantly made between the operational and the decisional world as to see that the principles which apply for the first (much better referenced) have nothing to do with the second. The most important aspect was the objective of efficient data warehouses which is quantified by the levels of user satisfaction and not by raw technical performance. From this, defining a series of service level indicators which quantify this subjective aspect is shown as the great challenge of DSS today. Service Level Agreements and Service Level Objectives are a direct part of these processes, and should always be taken into consideration with DSS management.

With Autonomic Computing we have approached the main problem that it is referenced with IT System management today: complexity. The 'enemy' of IT development is IT development itself and the requirements for semi-autonomic and autonomic solutions are on the order of day of any enterprise. To this end, IBM proposed the Autonomic Computing adoption model. The objective is clear and must not be mistaken with artificial intelligence (even if they are related): to help IT professionals focus on higher level tasks by leaving the low level tasks to autonomic computing managers. Technology and human work together, it is not about replacing the human expert but about helping him as he no longer can face on his own the challenges of complexity. Referring back to data warehouses, adopting autonomic computing for the operational and decisional world are completely different subjects. The idea was elaborated with the operational in mind, so in order to apply it for the decisional world we had to propose certain modifications (i.e. specific heuristics which take into account the DSS characteristics and service levels).

The AC model has at its core knowledge bases. This is where the web semantics and ontologies were taken into consideration for modeling the information. As the development of web towards the web 3.0 (semantic) is well emerging with recent years, using ontologies with AC seemed a valid approach. The state of the art presented some previous approaches towards this 'combination' (few to count) but with important results from our point of view,

which were in addition the starting point of this thesis. Moreover, ontologies have proven themselves lately and begin to be a viable alternative to the classical DBs. Information in the new era is seen as knowledge and should therefore be stored into knowledge bases. The efforts of big industry players such as Oracle are a testimony to the utilization of the ontology standards (such as RDF/OWL or SPARQL), which we consequently have adopted.

Contributions

Our main contributions come form the combination of the three domains itself. We wanted to offer a way of managing efficiently decision support systems, with all the information and knowledge required (from where the title of the thesis).

A ontology model for DSS

With data warehouses and decision support systems, the central contribution is a model of the DSS environment. Up to this point, no one was following the data warehouses, from where the high rate of failure with their implementations. It was 'cool' to do it, offered clear advantages but noone was following it. With the BI Copilot solutions, this monitoring is possible with the vision of an integrated CMDB. Starting from this, in the thesis we have proposed an ontology model of the DSS, with we have divided into three major parts: (i) the architecture, (ii) the configuration and performance indicators, and the (iii) pieces of advice and best practices for DSS management. These were formalized with OWL ontologies and ontology rules, which in turn were integrated as knowledge base for the autonomic computing model.

A autonomic computing adoption model for DSS

Second, by implementing autonomic computing with DSS, we have challenged the traditional ways in which AC works. The most important aspect was the non continuity in utilization of the data warehouses and the quality of service as the targeted performance indicator. We have adopted two heuristics which take into consideration the utilization periods (SLAs) and the QoS indicator, computed with regards to the SLAs. We have reinforced the fact that the QoS is not the same during utilization periods and we have proposed to compute the QoS as a scaled measure between all the elements that form the SLAs. To this end, we have implemented the first three AC principles (self-configuration, healing and optimization) with ACMs corresponding to each entity from the DSS architecture and to each AC purpose.

Experiments

The results we have presented with our approach were divided into three parts, each corresponding to one AC module: configuration, healing and optimization. In turn, each part contained two types of results: the ones obtained over theoretical environments, which we build especially for testing the approach and the ones obtained over real (client) environments which were used for validation.

- The self configuration experiments have shown that enterprises would benefit much more if their initial configuration of the data warehouses would take into consideration all the best practices and pieces of advice, specific for their

environment. Our approach benefits from the knowledge base that contains this information. The self configuration module affects the pre-production systems, so no real 'danger' of failure is present.

- Next, the BI Self-X self healing module experiments have shown the gain in both time and cost when performing diagnostics of data warehouses and on problem resolution. Whereas a DSS expert would take hours or days to find the problems and the solutions, the self healing module would perform the task in a matter of minutes (for the diagnostic). Moreover, enterprises using the BI Self-X self healing module can save substantial amounts of money, with the reduction of the intervention costs and of the non availability periods.
- The last series of experiments were performed for the self optimization module, which acted upon existing data warehouse configurations, following the rule: 'shoot first, ask the questions after'. This by itself is a high risk, but embedding the heuristics with very small thresholds allowed us to avoid general failures or errors. Even if sometimes drops in performance have been noticed, the system would achieve its goal of increasing the service levels and the user satisfaction. We have also proven that taking into account SLA/Os and expressing QoS based on them is much more important than looking at technical performances when speaking of optimization with data warehouses.

Returning to the aggregated view, the results we have obtained show two big advancements with DSS management: (i) monitoring the interesting information (BI Copilot) and (ii) usage of this information to configure, diagnostic, heal and optimize the DSS (BI Self-X). To our knowledge, there is no integrated solution today, neither in the world of academics or industry that performs these tasks.

9.2 Perspectives

Drawing the line is always hard, especially with emerging technologies. The perspectives are strongly related to our continuous work, beyond the elements that were presented with this thesis. We present our perspectives, from the research and the industrial point of view, as we have the privilege of interaction with both worlds:

Ontology – web semantics develops constantly, and at the time when this paragraph is written some of the features presented for OWL or SPARQL may have already become obsolete. This is a novel technology, and it is only now that it begins to be implemented in the industry more often. We consider that knowledge bases are the future of information systems for expressing knowledge. DBs are well referenced and highly used, but we see this new technology catching up fast. We are currently working on several projects that integrate the two (DBs & OWL), in which we try to make use of the benefits of the two technologies: expressiveness and complexity of the ontologies with the storage and fastness of the DBs.

The DSS Model - with the advancement of the BI Copilot CMDB solutions, the ontology model for BI Self-X has to evolve consequently. There is a continuous process of improvement. This means more advanced configurations, more configuration and performance indicators, an increased number of best practices etc. Software evolves each day, and if we want to be up to date, all new information must be integrated in the

knowledge bases. Therefore, the DSS ontology model should follow the guidelines for any ontology: it changes constantly and is the work of a collaborative effort.

Autonomic Computing we understand very well what are the 'en jeux' of this for future of IT and DSS. The ontology model provides us with all the information needed to manage the system. Autonomic computing plays the role of the executive, therefore from this perspective our AC model evolves constantly with the ontology model. As new diagnostics and problems occur, new actions are available. Also, we begin to step in the audit fields, as by watching the DSS audit experts at work, we are able to evolve the autonomic tasks from their conclusions and actions. The autonomic computing manager is therefore in a constant learning phase, as we constantly aliment it with knowledge.

As we mentioned earlier, several projects are under elaboration and development, based on the semantic technologies. An integration of all the items (advice, best practices and problems) that are mentioned in several sources (readme documents, technical forums etc.) is aimed with the SP2 BI Knowledge module. This makes use of both semantics capabilities and DB support. BI Knowledge uses ontologies of which contain information about types of software problems, types of possible actions, software functionalities and even a detailed software ontology that permits the identification of source, compatibility or migration aspects. The DBs come into play once all the information from the ontology is inferred, such that there is a fast access for all of the items. With the development of the BI software ontology, another project is part of our future work. We aim at building a detailed complete BI Software ontology with the help of the community, and for this we work on the TIMSys project. Its purpose is to allow users to define their DSS configurations and access them by a unique link, thus easing the task of system specifications.

The general perspectives for the development of semantic technologies are bright. Meta-data is everywhere and not integrating it would mean missing the next step in the evolution of the information systems. Maybe it will pass another 20 years before ontologies replace DBs, but this is a question we can't answer for now. The best we can do is see that the two are beginning to work together and fusion towards the future of knowledge bases. And, with them, the entire processes of IS management follows fast behind.

References

[Abelló *et al.*(2003)] Alberto Abelló, José Samos, and Félix Saltor. A data warehouse multidimensional data models classification. 2003.

[Ackoff (1967)] Russel L. Ackoff. Management misinformation systems. *Management Science*, 14:147–156, 1967.

[Agarwala *et al.*(2006)] Sandip Agarwala, Yuan Chen, Dejan Milojicic, and Karsten Schwan. Qmon: Qos-and utility-aware monitoring in enterprise systems. In *IEEE International Conference on Autonomic Computing*, pages 124–133, 2006.

[Agier *et al.*(2007)] Marie Agier, Jean-Marc Petit, and Einoshin Suzuki. Unifying framework for rule semantics: Application to gene expression data. *Fundam. Inf.*, 78:543–559, December 2007.

[Alesso and Smith (2006)] H. Peter Alesso and Craig F. Smith. *Thinking on the Web: Berners-Lee, Godel and Turing*. Wiley, John & Sons, Incorporated, 2006.

[A.L.I.C.E. (2010)] A.L.I.C.E. A.l.i.c.e. - the artifical intelligence internet computer entity, 2010. last accessed July 2010.

[Alliance (2005)] OSGi Alliance. About the ogsi service platform. Technical report, OSGi Alliance, 2005.

[Antoniou and Harmelen (2003)] Grigoris Antoniou and Frank Van Harmelen. Web ontology language: Owl. *Handbook on Ontologies in Information Systems*, pages 67–92, 2003.

[Aristotel (350 BCE)] Aristotel. *Metaphysics*. 350 B.C.E.

[Astah (2010)] Astah. Astah* uml, 2010. last accessed decembre 2010.

[Badger and Todd Hughes (2004)] Lee Badger and DARPA Todd Hughes. Tolerant systems - self-regenerative systems, 2004. last accessed July 2010.

[Badiru and Cheung (2002)] Adedeji Bodunde Badiru and John Cheung. *Fuzzy Engineering Expert Systems With Neural Network Applications*. John Wiley & Sons Inc, 2002.

[Banker and Kauffman (2004)] Rajiv D. Banker and Robert J. Kauffman. The evolution of research on information systems: A fiftieth-year survey of the literature in management science. *Management Science*, 50(3):281–298, March 2004.

[Bell (2003)] Donald Bell. Uml basics: An introduction to the unified modeling language, 2003. last access july 2010.

[Belleil (2009)] Claude Belleil. *Le langage UML 2.0*. Université de Nantes, 2009.

[Berners-Lee (1989)] Tim Berners-Lee. Information management: A proposal. *CERN*, 1989.

[Berners-Lee (2006)] Tim Berners-Lee. Linked data - design issues, 2006. last accessed July 2010.

[Berson and Smith (1996)] Alex Berson and Stephen J. Smith. *Data Warehousing, Data Mining, and Olap, 1st edition*. McGraw-Hill, Inc, 1996.

[Biazetti and Gajda (2005)] Ana Biazetti and Kim Gajda. Achieving complex event processing with active correlation technology, 2005. last accessed septemeber 2010.

[Bizer *et al.*(2009a)] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[Bizer *et al.*(2009b)] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia – a crystallization point for the web of data. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 7:154–165, 2009.

[Boreisha and Myronovych (2007)] Yuri Boreisha and Oksana Myronovych. Web-based decision support systems as knowledge repositories for knowledge management systems. In *The 2007 International Conference on Information and Knowledge Engineering*, 2007.

[Bouaud *et al.*(1995)] J. Bouaud, B. Bachimont, J. Charlet, and P. Zweigenbaum. Methodological principles for structuring an "ontology". In *IJCAI'95 Workshop on "Basic Ontological Issues In Knowlege Sharing"*, 1995.

[Bozsak *et al.*(2002)] Erol Bozsak, Marcc Ehrig, Siegfried Handschuh, Siegfried H, Alexander Maedche, Andreas Hotho, Er Maedche, Boris Motik, Daniel Oberle, Christoph Schmitz, Nenad Stojanovic, Rudi, Steffen Staab, Ljiljana Stojanovic, and Valentin Zacharias. Kaon - towards a large scale semantic web. In *The Third International Conference on E-Commerce and Web Technologies*, pages 304–313, 2002.

[Brynjolfsson and Hitt (1996)] Erik Brynjolfsson and Lorin Hitt. Paradox lost? firm-level evidence on the returns to information systems spending. *Management Science*, 42(4):541–558, 1996.

[Builders (2010)] Information Builders. Decision support systems – dss, 2010. last accessed july 2010.

[Bush (1945)] Vannevar Bush. As we may think. *The Atlantic Monthly*, 1945.

[Bézivin *et al.*(2003)] Jean Bézivin, Sébastien Gérard, Pierre-Alain Muller, and Laurent Rioux. Mda components: Challenges and opportunities. In *First International Workshop Metamodelling for MDA*, 2003.

[Centre (2010)] Business Intelligence Centre. Extract transform load (etl), 2010. last accessed july 2010.

[CERN (2010)] CERN. Eurpean organization for nucelar research, 2010. last accessed July 2010.

[Chaudhuri and Dayal (1997)] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Rec.*, 26(1):65–74, 1997.

[Chaudhuri *et al.*(2001)] Surajit Chaudhuri, Umeshwar Dayal, and Venkatesh Ganti. Database technology for decision support systems. *Computer*, 34(12):48–55, 2001.

[CHISEL (2008)] CHISEL. Jambalaya, 2008. last accessed July 2010.

[Codd *et al.*(1993)] Edgar F. Codd, S.B. Codd, and C.T. Salley. Providing olap to user-analysts: An it mandate. 1993.

[Codd (1970)] E. F. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13(6):370–387, 1970.

[ComputerAssociates (2005)] CA ComputerAssociates. The adoption of itil in large enterprises. Technical report, Tech Republic, 2005.

[Courtney (2001)] James F. Courtney. Decision making and knowledge management in inquiring organizations: toward a new decision-making paradigm for dss. *Decision Support Systems*, 31(1):17–38, 2001.

[Daconta *et al.*(2003)] Michael C. Daconta, Leo J. Obrst, and Kevin T. Smith. *The Semantic Web: A Guide to the Future of XML, Web Services and Knowledge Management*. John WIley & Sons, Inc., 2003.

[DARPA (2009)] DARPA. Defense advanced research project agency, 2009. last accessed July 2010.

[Das (2009)] Souripriya Das. Rdf support in oracle rdbms. Technical report, Oracle New England Development Center, 2009.

[Davis (2006)] Mills Davis. Semantic wave 2006 - executive guide to the business value of semantic technologies, 2006. last accessed July 2010.

[Dekkers and Weibel (2003)] Makx Dekkers and Stuart Weibel. State of the dublin core metadata initiative. *D-Lib Magazine*, 9(4), 2003.

[Dell (1989)] Michael Dell. The connected economy. *Information Resources Management Journal*, 2(3):1–13, 1989.

[Delteil *et al.*(2001)] Alexandre Delteil, Catherine Faron-Zucker, and Rose Dieng. Extension of rdfs based on the cgs formalisms. In *In preceedings of the 9th International Conference on Conceptual Structures - ICCS'01*, pages 275–289, 2001.

[Ding *et al.*(2005)] Li Ding, Pranam Kolari, Zhongli Ding, Sasikanth Avancha, Tim Finin, and Anupam Joshi. Using ontologies in the semantic web: A survey. Technical report, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 2005.

[DMTF (2010)] DMTF. Common information model, 2010. last accessed septemeber 2010.

[Dowling *et al.*(2006)] Jim Dowling, Raymond Cunningham, Eoin Curran, and Vinny Cahill. Building autonomic systems using collaborative reinforcement learning. *The Knowledge Engineering Review*, 21(3):231–238, 2006.

[Dumont (2007)] Christian Dumont. *ITIL pour un service informatique optimal 2e édition*. Eyrolles, 2007.

[Elnaffar *et al.*(2003)] Said Elnaffar, Wendy Powley, Darcy Benoit, and Pat Martin. Today's dbmss: How autonomic are they? In *14th International Workshop on Database and Expert Systems Applications*, 2003.

[ETLtool.com (2010)] ETLtool.com. Etl tool survey, 2010. last accessed july 2010.

[Farber (2007)] Dan Farber. The semantic wedge: Freebase, powerset and twine, 2007. last accessed July 2010.

[Fensel *et al.*(2001)] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Deborah L. McGuinness, and Peter F. Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[Freebase (last accesed 2010)] Freebase. Freebase - api reference, last accesed 2010.

[Ganek and Corbi (2003)] Alan G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 43(1):5–18, 2003.

[Garlan and Schmerl (2002)] David Garlan and Bradley Schmerl. Model-based adaptation for self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 27–32, 2002.

[Globerson *et al.*(1991)] Arye Globerson, Shlomo Globerson, and Judith Frampton. *You Can't Manage What You Don't Measure: Control and Evaluation in Organizations*. Avebury, 1991.

[Gonzales *et al.*(2007)] Juan M. Gonzales, Jose A. Lozano, Jorge E. Lopez de Vergara, and Victor A. Villagra. Self-adapted service offering for residential environments. In *1st IEEE Workshop on Autonomic Communications and Network Management, ACNM'07*, pages 48–55, 2007.

[Google (2010)] Google. Google, 2010. last accessed July 2010.

[Gorry and Morton (1971)] George Anthony Gorry and Michael S. Scott Morton. *A Framework for Management Information Systems*. 1971.

[Grabova *et al.*(2010)] Oksana Grabova, Jérôme Darmont, Jean-Hugues Chauchat, and Iryna Zolotaryova. Business intelligence for small and middle-sized entreprises. *SIGMOD Record*, 39:39–50, 2010.

[Grau *et al.*(2008)] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008.

[Grembergen (2003)] Wim Van Grembergen. Introduction to the minitrack it governance and its mechanisms. In *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, volume 8, page 242, 2003.

[Grobe (2009)] Michael Grobe. Rdf, jena, sparql and the "semantic web". In *SIGUCCS '09: Proceedings of the ACM SIGUCCS fall conference on User services conference*, pages 131–138, 2009.

[Groison (2001)] David Groison. En quête de pertinence. *Science et vie*, (652):120–122, 2001.

[Group (2010)] Object Modeling Group. Omg model driven architecture, 2010. last accessed july 2010.

[Group (2011)] The Integrated Solution Group. The data warehouse data flow, 2011. last accessed - Janvier 2011.

[Gruber (1992)] Tom Gruber. *What is an ontology?* Academic Press Pub., 1992.

[Gruber (1993a)] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Formal Ontology in Conceptual Analysis and Knowledge Representation*, 1993.

[Gruber (1993b)] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[Guo (2003)] H. Guo. A bayesian approach for autonomic algorithm selection. In *In Proceedings of the IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, 2003.

[Haarslev and Möller (2001)] Volker Haarslev and Ralf Möller. Racer system description. In *Preceedings of the First Interntation Joint Conference on Automated Reasoning*, pages 701–706, 2001.

[Haase *et al.*(2005)] Peter Haase, , Peter Haase, and Ljiljana Stojanovic. Consistent evolution of owl ontologies. In *ESWC 2005*, pages 182–197, 2005.

[Hakia (2010)] Hakia. Hakia, 2010. last accessed July 2010.

[Hert *et al.*(2010)] Matthias Hert, Gerald Reif, and Harald C. Gall. Updating relational data via sparql/update. In *EDBT Workshop*, 2010.

[Holsapple and Whinston (1996)] Clyde W. Holsapple and Andrew B. Whinston. *Decision Support Systems: A Knowledge Based Approach*. West Group, 1996.

[Horn (2001)] Paul Horn. Autonomic computing: Ibm's perspective on the state of information technology. 2001.

[Horrocks *et al.*(2003)] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From shiq and rdf to owl: the making of a web ontology language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7–26, 2003.

[Horrocks (1999)] Ian Horrocks. Fact and ifact. In *Proceedings of the International Workshop on Description Logics DL'99*, pages 133–135, 1999.

[Huebscher and McCann (2008)] M.C. Huebscher and J.A. McCann. A survey on autonomic computing – degrees, models and applications. *ACM Computing Surveys*, 40(3):1–28, 2008.

[Iannone and Rector (2008)] Luigi Iannone and Alan Rector. Calculations in owl. In *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, OWLED 2008*, volume 432, 2008.

[IBM (2001)] IBM. *An architectural blueprint for autonomic computing*. IBM Corporation, 2001.

[IBM (2004)] IBM. Yourturn—global ceo study 2004. survey of 456 ceos worldwide., 2004. last accessed July 2010.

[IBM (2005a)] IBM. Understand web services distributed management (wsdm), 2005. last accessed July 2010.

[IBM (2005b)] Corporation IBM. Autonomic computing. powering your business for success. *International Journal of Computer Science and Network Security*, 7(10):2–4, 2005.

[IBM (2006)] IBM. *An architectural blueprint for autonomic computing, 4th Edition*. IBM Corporation, 2006.

[IBM (2008)] OASIS & IBM. Oasis solution deployment descriptor (sdd), 2008. last accessed July 2010.

[IBM (2010)] IBM. Tivoli software library for technical resources, 2010. Last accessed - December 2010.

[Imhoff (1999)] Claudia Imhoff. The corporate information factory, 1999. last accessed july 2010.

[Inmon *et al.*(1999)] William H. Inmon, Ken Rudin, Christopher K. Buss, and Ryan Sousa. *Data Warehouse Performance*. John WIley & Sons, Inc., 1999.

[Inmon (1996)] William H. Inmon. The data warehouse and data mining. *Communications of the ACM*, 39(11):49–50, 1996.

[Inmon (1999)] Bill Inmon. Data mart does not equal data warehouse, November 1999. last accessed july 2010.

[Inmon (2005)] William H. Inmon. *Building the data warehouse, fourth edition*. Wiley Publishing, 2005.

[Inmon (2010a)] Bill Inmon. Corporate information factory, 2010. last accesed july 2010.

[Inmon (2010b)] Bill Inmon. Data warehousing: Kimball vs. inmon (by inmon), July 2010. last accessed july 2010.

[Inmon (2010c)] Bill Inmon. The government information factory, 2010. last accesed july 2010.

[ISACA (2010)] ISACA. Cobit 5 design (exposure april 2010), 2010. last accessed july 2010.

[Janus and Fouché (2009)] Philo Janus and Guy Fouché. Introduction to olap. *Pro SQL Server 2008 Analysis Services*, pages 1–14, 2009.

[Jena (2010)] Jena. Jena 2 inference support, 2010. last accessed July 2010.

[Keen and Morton (1978)] Peter G. W. Keen and Michael S. Scott Morton. *Decision Support Systems: An Organizational Perspective*. Addison-Wesley Publishing, 1978.

[Kimball (1996)] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, 1996.

[Klein (2005)] Allyson Klein. Optimizing enterprise it. *Intelligent Enterprise*, 2005.

[Krivov *et al.*(2007)] Serguei Krivov, Ferdinando Villa, Richard Williams, and Xindong Wu. *On Visualization of OWL Ontologies - Chapter 10 from Semantic Web*. Springer US, 2007.

[Krotzsch *et al.*(2008)] Markus Krotzsch, Sebastian Rudolph, and Pascal Hitzler. Elp: Tractable rules for owl 2. In *International Semantic Web Conference ISWC'08*, 2008.

[Lassila and McGuinness (2001)] Ora Lassila and Deborah McGuinness. The role of frame-based representation on the semantic web. Technical report, Knowledge Systems Laboratory, Stanford University and Software Technology Laboratory, Nokia Research Center, 2001.

[Lewis and Ray (1999)] Lundy Lewis and Pradeep Ray. Service level management definition, architecture, and research challenges. In *Global Telecommunications Conference, 1999. GLOBECOM '99*, pages 1974 – 1978, 1999.

[Liang *et al.*(2005)] Yaozhong Liang, Harith Alani, and Nigel Shadbolt. Ontology change management in protege. In *In Proceedings of the 1st AKT Doctoral Symposium, Milton Keynes*, 2005.

[Lightstone *et al.*(2003)] Sam Lightstone, Berni Schiefer, Danny Zilio, and Jim Kleewein. Autonomic computing for relational databases: the ten-year vision. In *Proceedings of the IEEE Workshop Autonomic Computing Principles and Architectures (AUCOPA'03)*, pages 419–424, 2003.

[Linnaeus (1731)] Carl Linnaeus. Taxonomy, 1731. last accessed July 2010.

[Littman *et al.*(2004)] M.L. Littman, N. Ravi, Eitan Fenson, and Rich Howard. Reinforcement learning for autonomic network repair. In *Proceedings of the 1st International Conference on Autonomic Computing*, pages 284–285, 2004.

[Liu and Özsu (2008)] Ling Liu and M. Tamer Özsu. *Encyclopedia of Database Systems*. Springer-Verlag, 2008.

[Lupu and Solman (1997)] Emil Lupu and Morris Solman. Conflict analysis for management policies. In *Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management*, pages 430–443, 1997.

[Lymberopoulos *et al.*(2003)] Leonidas Lymberopoulos, Emil Lupu, and Morris Sloman. An adaptive policy-based framework for network services management. *Journal of Network and Systems Management*, 11(3):277–303, 2003.

[Maedche *et al.*(2003)] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer, and Raphael Volz. Ontologies for enterprise knowledge management. *IEEE Intelligent Systems*, 18(2):26–33, 2003.

[Magee *et al.*(1995)] Jeff Magee, Naranker Dulay, and Susan Eisenbachand Jeff Kramer. Specifying distributed software architectures. *Software Engineering - ESEC 95*, 989:137–153, 1995.

[Mailvaganam (2007a)] Hari Mailvaganam. Future of datamining - from obscurity to center stage, 2007. last accessed july 2010.

[Mailvaganam (2007b)] Hari Mailvaganam. Introduction to olap - slice, dice and drill!, 2007. last accessed july 2010.

[Maisons (2006)] David Maisons. Datawarehouse et datamining. Technical report, Conservatoire Regionnal des Art et Metiers Centre de Versailles, 2006.

[Markl *et al.*(2003)] Volker Markl, Guy M. Lohman, and Vijayshanka Raman. Leo : An autonomic optimizer for db2. *IBM Systems Journal*, 42(1):98–106, 2003.

[Mateen *et al.*(2008)] Abdul Mateen, Basid Raza, and Tauqeer Hussain. Autonomic computing in sql server. In *Proceedings of the 7th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2008*, pages 113–118, 2008.

[Mazon *et al.*(2005)] Jose-Norberto Mazon, Juan Trujillo, Manuel Serrano, and Mario Piattini. Applying mda to the development of data warehouses. In *The 8th ACM international workshop on Data warehousing and OLAP*, pages 57–66, 2005.

[Mika *et al.*(2005)] Peter Mika, Michel Klein, and Radu Serban. Semantics-based publication management using rss and foaf. In *Proceedings of the 1st Workshop on the Semantic Desktop (SD 2005), 4th International Semantic Web Conference*, 2005.

[Miller (2008)] Brent Miller. The autonomic computing edge: Can you chop up autonomic computing?, 2008. last accessed septmeber 2010.

[Miller (2010)] George A. Miller. Wordnet, 2010. last accessed July 2010.

[MindFusion (2010)] MindFusion. Xmll viewer 3.0, 2010. last accessed september 2010.

[Motik *et al.*(2002)] Boris Motik, Alexander Maedche, and Raphael Volz. A conceptual modeling approach for semantics-driven enterprise applications. In *First International Conference Ontologies, Databases and Application of Semantics (ODBASE 2002)*, pages 1082–1099, 2002.

[Motika *et al.*(2005)] Boris Motika, Ulrike Sattlerb, and Rudi Studera. Query answering for owl-dl with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.

[Mozes (1989)] E. Mozes. A deductive database based on aristotelian logic. *Journal of Symbolic Computation*, 7(5):487–507, 1989.

[Muscettola *et al.*(1998)] Nicola Muscettola, Pandurang P. Nayak, Barney Pell, and Brian C. Williams. Remote agent: to boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[Necib and Freytag (2005)] Chokri Ben Necib and Johann-Christoph Freytag. Query processing using ontologies. In *International conference on advanced information systems engineering. CAISE 2005*, volume 3520, pages 167–186, 2005.

[Nelson and Wright (2005)] Greg Barnes Nelson and Jeff Wright. Real time decision support: Creating a flexible architecture for real time analytics, 2005. last accessed july 2010.

[Nelson (2010)] Ted Nelson. Who invented hypertext, web history, 2010. last accessed July 2010.

[NetworkWorkingGroup (1999)] NetworkWorkingGroup. Hypertext transfer protocol – http/1.1, 1999. last accessed July 2010.

[NetworkWorkingGroup (2005)] NetworkWorkingGroup. Uniform resource identifier (uri): Generic syntax, 2005. last accessed July 2010.

[Normand (2007)] Cristophe Normand. Le web sémantique appliqué au calcule autonomique. Memoire cnam, Ecole Polytechnique de l'Université de Nantes, 2007.

[Noy and McGuinness (2001)] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology, 2001. last accessed July 2010.

[Oasis (2008)] Oasis. Technology report - dmtf common information model (cim), 2008. last accessed 2010.

[Object Services & Consulting (2002)] Inc. Object Services & Consulting. Probemeister, 2002. last accessed July 2010.

[O'Brien (2002)] James A. O'Brien. *Introduction to Information Systems: Essentials for the E-Business Enterprise*. McGraw-Hill, 2002.

[O'Connor *et al*.(2008)] Martin O'Connor, Csongor Nyulas, Ravi Shankar, Amar Das, and Mark Musen. The swrlapi: A development environment for working with swrl rules. In *Proceedings of the International Workshop on OWL: Experiences and Directions, OWLED 2008.*, 2008.

[O'Connor (2008)]Martin O'Connor. Swrltab a development environment for working with swrl rules in protégé-owl. Technical report, Standford Medical Informatics, Standford University, 2008.

[of Medicine University of Standford (2010)] School of Medicine University of Standford. The portégé editor and knowledge acquisition system, 2010.

[Oracle (2008)] Oracle. Management excellence: The metrics reloaded. Technical report, Oracle, 2008.

[Oracle (2009)] Oracle. Oracle semantic technologies inference best practices with rdfs/owl - whitepaper. Technical report, Oracle, 2009.

[Oracle (2010)] Oracle. The multidimensional data model, 2010. last accessed july 2010.

[Osogami *et al*.(2005)] Takayuki Osogami, Mor Harchol-Balter, and Alan Scheller-Wolf. Analysis of cycle stealing with switching times and thresholds. *Performance Evaluation*, 61(4):347–369, 2005.

[Parshar and Hariri (2007)] Manish Parshar and Salim Hariri. *Autonomic Computing: Concepts, Infrastructure and Applications*. CRC Press, Taylor & Francis Group, 2007.

[Patel-Schneider (2007)] Peter F. Patel-Schneider. Safe rules for owl 1.1. Technical report, Bell Labs Research, Alcatel-Lucent, 2007.

[Pender (2003)] Tom Pender. *UML Bible*. Wiley, 2003.

[Pidcock (2003)] Woody Pidcock. What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta-model?, January 2003. last accessed July 2010.

[PowerSet (2010)] PowerSet. Powerset, 2010. last accessed July 2010.

[Pretorius (2004)] A. Johannes Pretorius. Ontologies - introduction and overview. 2004.

[Raimbault (2008)] Thomas Raimbault. *Transition de Modèles de connaissances*. PhD thesis, LINA, 2008.

[Rector and Stevens (2008)] Alan Rector and Robert Stevens. Barriers to the use of owl in knowledge driven applications. In *Fifth International workshop on OWL Experiences and Directions 2008, OWLED 2008*, 2008.

[Ridley *et al.*(2004)] Gail Ridley, Judy Young, and Peter Carroll. Cobit and its utilization: a framework from the literature. In *The 37th Annual Hawaii International Conference on System Sciences*, 2004.

[R.N.Anthony (1965)] R.N.Anthony. *Planning and Control Systems: A Framework for Analysis*. PhD thesis, Harvard University Graduate School of Business Administration, Cambridge, MA, 1965.

[SAP (2010)] SAP. Data, data everywhere: A special report on managing information. Technical report, SAP America, Inc., 2010.

[Segaran *et al.*(2009)] Toby Segaran, Colin Evans, and Jamie Taylor. *Programming the Semantic Web*. O'Reilly Media, 2009.

[Selectorweb (2010)] Selectorweb. Etl - extract, transform, load, 2010. last accessed 2010.

[Shadbolt *et al.*(2006)] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The semantic web revisited. *Intelligent Systems*, 21(3):96–101, 2006.

[Shim *et al.*(2002)] J.P. Shim, Merrill Warkentin, James F. Courtney, Daniel J. Power, Ramesh Sharda, and Christer Carlsson. Past, present, and future of decision support technology. *Decision Support Systems*, 33(2):111–126, 2002.

[Simon (1977)] Herbert Alexander Simon. *The New Science of Management Decision*. Prentice Hall PTR, 1977.

[Sirin and Parsia (2007)] Evren Sirin and Bijan Parsia. Sparql-dl: Sparql query for owl-dl. In *3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.

[Sirin *et al.*(2007)] Evren Sirin, Bijan Grau, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.

[SP2Solutions (2010)] SP2Solutions. Sp2 solutions homepage, 2010.

[Staab and Maedche (2000)] Steffen Staab and Alexander Maedche. Axioms are objects, too - ontology engineering beyond the modeling of concepts and relations. In *Proceedings of the Workshop of Applications of Ontologies and Problem-solving Methods, ECAI 2000*, 2000.

[Stojanovic *et al.*(2004)] Ljiljana Stojanovic, Jurgen M. Schneider, Alexander D. Maedche, S Libischer, Rudi Studer, Th. Lumpp, Andreas Abecker, G. Breiter, and J. Dinger. The role of ontologies in autonomic computing systems. *IBM Systems Journal*, 43(3):598–616, 2004.

[Stuckenschmidt (2008)] Heiner Stuckenschmidt. Debugging owl ontologies - a reality check. In *The 6th International Workshop on Evoaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON-SWSC-2008)*, 2008.

[StylusInc (2008)] StylusInc. Software development life cycle (sdlc), 2008. last accessed july 2010.

[Sutton and Bart (1998)] Richard S. Sutton and Andrew G. Bart. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, 1998.

[Swoogle (2010)] Swoogle. Swoogle, 2010. last accessed July 2010.

[Technology (2004)] Apogee Communications Technology. Model driven architecture, an introduction. Technical report, 2004.

[Tesauro *et al.*(2004)] Gerald Tesauro, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, and Steve R. White. A multi-agent systems approach to autonomic computing. In *The Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '04*, pages 464–471, 2004.

[Tesauro *et al.*(2006)] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Preceedings of the 5th IEEE International Conference on Autonomic Computing, ICAC'06*, pages 65–73, 2006.

[Toffler (1991)] A. Toffler. *Powershift: Knowledge, Wealth and Power at the edge of the 21st Century*. Bantam Book Publishing, 1991.

[TopQuadrant (2004)] TopQuadrant. Virtuoso: A first impression. *Business Integration Journal*, 2004.

[TopQuadrant (2009)] TopQuadrant. Semantic web solutions at work in the enterprise. Technical report, TopQuadrant, 2009.

[tpc (2010)] Tpc - transaction processing performance council, 2010. last accessed July 2010.

[Triou *et al.*(2007)] Frédéric Triou, Fabien Picarougne, and Henri Briand. Apport du web sémantique dans la réalisation d'un moteur de recherche géo-localisé à usage des entreprises. In *In preceedings of EGC 2007*, 2007.

[Tsarkov and Horrocks (2006)] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2006)*, pages 292–297, 2006.

[Tudorache (2007)] Tania Tudorache. Collaborative ontology development in protégé. Technical report, Standford University, 2007.

[Vassiliadis *et al.*(1999)] Panos Vassiliadis, Mokrane Bouzeghoub, and Christoph Quix. Towards quality-oriented data warehouse usage and evolution. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering, CAISE 99*, pages 164–179, 1999.

[Vrandecic *et al.*(2010)] Denny Vrandecic, Markus Krotzsch, Sebastian Rudolph, and Uta Losch. Leveraging non-lexcial knowledge for the linked open data web. *The Fith RAFT 2010*, 1:18–27, 2010.

[W3C (2010a)] W3C. Owl 2 web ontology language: Profiles, 2010. last accessed July 2010.

[W3C (2010b)] W3C. Owl web ontology language current status, 2010. last accessed July 2010.

[W3C (2010c)] W3C. Rdf vocabulary description language 1.0: Rdf schema, 2010. last accessed July 2010.

[W3C (2010d)] W3C. Rif in rdf, 2010. last accessed July 2010.

[W3C (2010e)] W3C. World wide web consortium, 2010. last accessed July 2010.

[W3C (2010f)] W3C. Xml essentials, 2010. last accessed July 2010.

[W3C (2010g)] W3C. Xml query (xquery), 2010. last accessed July 2010.

[W3Schools (2010)] W3Schools. Soap introduction, 2010. last accessed July 2010.

[Wang *et al.*(2004)] Eric Wang, Sung Ah Kim, and Yong Se Kim. Case study in rule inferencing using protégé and jess. Technical report, Creative Design Intelligent Tutoring Systems (CreDits) Research Center, Sungkyunkwan University, 2004.

[Wang *et al.*(2005)] Xiaoshu Wang, Robert Gorlitsky, and Jonas S. Almeida. From xml to rdf: how semantic web technologies will change the design of 'omic' standards. *Nature Biotechnology*, 23:1099–1103, 2005.

[Weill and Ross (2004)] Peter Weill and Jeanne Ross. *IT Governance: How Top Performers Manage IT Decision Rights for Superior Results*. Harvard Business Press, 2004.

[Wooldridge and Jennings (1995)] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[Wooldridge (2009)] Michael Wooldridge. *An Introduction to MultiAgent Systems - Second Edition*. John Wiley & Sons, 2009.

[WorldWideWebSize.com (2010)] WorldWideWebSize.com. Daily estimated size of the world wide web, 2010. last accessed July 2010.

[Wu (2007)] Zhe Wu. Oracle: The semantic web for application developers, 2007. last accessed July 2010.

[Zablith (2009)] Fouad Zablith. Ontology evolution:a practical approach. Technical report, Knowledge Media Institute (KMi), The Open University, 2009.

[Zhang and Figueiredo (2006)] Jian Zhang and R.J. Figueiredo. Autonomic feature selection for application classification. In *IEEE International Conference on Autonomic Computing, ICAC '06*, pages 43–52, 2006.

Résumé:

Les travaux de cette thèse combinent trois domaines de recherche : (i) la gestion des Systèmes d'Information Décisionnel (SID) et les entrepôts de données, (ii) la gestion autonomique avec le Calcul Autonomique et (iii) l'intégration des connaissances avec les technologies sémantiques et les ontologies.
Dans la littérature, la plupart des travaux traitent les Systèmes Opérationnels, fondamentalement différent des SID. Les SID manquent de pratiques bien définies pour leur gestion. Dans ce contexte, la thèse adresse deux problématiques : (i) l'intégration des connaissances pour la gestion des SID à l'aide des ontologies et (ii) l'utilisation du Calcul Autonomique en tenant compte des particularités des SID.
Les apports principaux de cette thèse sont :(i) l'élaboration d'une ontologie qui modélise le SID et sa gestion, comprenant donc : l'architecture des entrepôts de données, les paramètres et les performances subjectives (Qualité des Services), ainsi que les conseils de gestion; (ii) l'élaboration d'un modèle de Calcul Autonomique permettant au SID d'assurer des fonctions d'autogestion : configuration, diagnostic/réparation et optimisation, avec le but d'améliorer les niveaux de service ; (iii) le développement de l'approche BI Self-X, composée de trois modules, chacun chargé d'une fonction de gestion CA. Les résultats obtenus avec cette approche ont montré que les entreprises qui utilisent BI Self-X pour la gestion de leur SID ont des meilleures performances, ainsi qu'une baisse des coûts et du temps passé dans l'implémentation et la maintenance de leurs entrepôts de données.

Mots clés : Système d'Information Décisionnel, Entrepôt de Données, Ontologie, Calcul Autonomique, Qualité de Service

Abstract:

This thesis combines three major research domains: (i) the management Decision Support Systems (DSS) and Data Warehouses, (ii) autonomic task management using Autonomic Computing and (iii) the transformation and modeling of knowledge by adopting Web Semantic technologies and Ontologies.
In the literature, most of the references are done towards Operational Systems, which are fundamentally different from DSSs. There is a lack of well defined management best practices for DSS. In this context the two main issues are addressed: (i) the integration of the DSS management knowledge into a unified knowledge source with the help of ontologies and (ii) the usage of the integrated knowledge base with the Autonomic Computing model.
The principal contributions of the thesis are: (i) the elaboration of an ontology model of the DSS and its management policies, which includes architectures, parameters, technical performances, subjective performances (QoS), best practices, known issues, service levels (SLA/O); (ii) the elaboration of an autonomic computing adoption model that provides the DSS with self management functions: configuration, healing and optimization, with the main purpose of improving the levels of service; (iii) the development of BI Self-X, composed of three modules each in charge of an AC self management function. The results obtained with this approach have proven that enterprises using BI Self-X with their DSS have increased performance and service levels while decreasing the costs and time in the implementation and maintaining of their data warehouses.

Keywords: Decision Support System, Data Warehouse, Ontology, Autonomic Computing, Service Level Agreements, Quality of Service