ONTOLOGY-BASED AUTONOMIC COMPUTING FOR RESOURCE SHARING BETWEEN DATA WAREHOUSES IN DECISION SUPPORT SYSTEMS

Vlad Nicolicin-Georgescu*, Vincent Benatier
SP2 Solutions, 85000 La Roche sur Yon, France

Remi Lehn, Henri Briand
**LINA CNRS 6241, COD Team, Ecole Polytechnique de l'Unviersité de Nantes, Nantes, France*

Keywords: Autonomic Computing, Decision Support System, Data Warehouse, Ontology.

Abstract: Complexity is the biggest challenge in managing information systems today, because of the continuous growth in data and information. As decision experts, we are faced with the problems generated by managing Decision Support Systems, one of which is the efficient allocation of shared resources. In this paper, we propose a solution for improving the allocation of shared resources between groups of data warehouses within a decision support system, with the Service Levels Agreements and Quality of Service as performance objectives. We base our proposal on the notions of autonomic computing, by challenging the traditional way of autonomic systems and by taking into consideration decision support systems' special characteristics such as usage discontinuity or service level specifications. To this end, we propose the usage of specific heuristics for the autonomic self-improvement and integrate aspects of semantic web and ontology engineering as information source for knowledge base representation, while providing a critical view over the advantages and disadvantages of such a solution.

1 INTRODUCTION

With the fast increasing both in size and complexity of analytical data, improving data warehouse performances is becoming a major challenge in the management of information systems. It has been shown that between 70 and 90 percent of the enterprises consider their data warehousing efforts inefficient at the end of the first year (Frolick & Lindsey 2003). In most cases, the causes are elevated costs and inadequate management. Indeed, decisional experts spend a lot of time on low level tasks, such as resource configuration, at the expense of high-level objectives, such as reporting or business policy adoption.

Autonomic Computing (AC), introduced in 2001 by IBM (IBM 2001), seems a promising way to overcome such difficulties. The initial objective was to introduce self-configuration, self-optimization, self-healing and self-protection in IT systems. The operationalization in the IBM model is supported by a so-called Autonomic Computing Manager (ACM), which is based on a closed four phase loop (monitor, analyze, plan and execute) around a central knowledge-base (MAPE-K loop). Several recent industrial tentative have been proposed, in the context of data-base management. In (Markl et al. 2003), the authors proposed LEO as a software agent for improving DB2 database queries. (Mateen et al. 2008) presented how Microsoft SQL Server's can enable AC through several specific modules, but only objective specific ones and not for the entire system. In (Lightstone et al. 2002), another proposed approach for DB2 adoption of AC presents how self-management can be used for reducing task complexity by several advisors: serviceability utility, configuration advisor, design advisor and query optimizer.

It is now well-known that the efficiency of AC adoption is closely linked to the quality of the knowledge-base. With the expansion of the semantic Web, ontologies have become a standard to model complex informational systems. From the seminal work of (Maedche et al. 2003), introducing

Nicolicin-Georgescu V., Benatier V., Lehn R. and Briand H. (2010).
ONTOLOGY-BASED AUTONOMIC COMPUTING FOR RESOURCE SHARING BETWEEN DATA WAREHOUSES IN DECISION SUPPORT SYSTEMS.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 199-206
DOI: 10.5220/0002895501990206

ontologies as knowledge-base formalization for the autonomic manager has been proposed by different authors ((Nicolicin-Georgescu et al. 2009), (Stojanovic et al. 2004)). The semantic approach allows to unify different forms of information (such as expert knowledge, advice and practices from readme documents, technical forums, etc.) by relying on increased expressivity and the reasoning capabilities offered by reasoning engines such as those presented by (Sirin et al. 2007).

In this paper we address a very important problem in Decision Support Systems (DSS) management: the allocation of cache memories between groups of Data Warehouses (DWs) sharing the same amount of common RAM memory. As DW sizes are very large (up to hundreds of TB), operations such as information retrieval or aggregation are very time consuming. To resolve this problem, caches are used for storing a part of the most frequently used data into RAM, so that it can be accessed faster. As the quantity of installed RAM is limited by either cost or, more often, platform, an efficient and dynamic allocation is required. Despite its crucial role in the performances of data warehouses, this low level repetitive task is generally done manually, which implies expert time wasting and human errors. And, as far as we know this problem remains open and it has drawn few attention in the DSS literature.

Thus, in this paper we propose an innovative approach, which adopts ontology-based AC over the specific characteristics of decision support systems. There are two innovative aspects. The first consists in the application of the two technologies (AC and Web Semantics) for performance improvement (self-optimization), whereas literature has treated mostly the aspect of problem resolution (self-healing). The second is its application over DSSs that have special characteristics such as usage discontinuity and usage purpose, in comparison to the classical approaches of operational systems.

The remainder of the paper is organized as follows. The Section 2 presents the architecture of the DSS, with limits of such systems and our objectives. It also introduces the usage of ontologies with a modelling based on autonomic adoption policies. Section 3 focuses on the adoption of the autonomic manager MAPE-K loop for DSS. It introduces two proposed heuristics with the help of ontology based rules, with the purpose of (autonomic) managing cache allocations within data warehouses. Section 4 introduces the experiments performed, describing the test protocol and the results obtained with this approach. Finally, in the Section 5 we give the conclusion and the future directions for our work.

2 THE ARCHITECTURE OF THE DSS MANAGEMENT MODEL

This section introduces the aspects of modeling DSS for AC adoption. We discuss first the limits of AC adoption and our objectives regarding these limits. Then we present the architectural modeling of a DSS, introducing also our proposition related to the mentioned objectives.

2.1 Limits and Objectives

AC adoption over a DSS is faced with several important limits, such as: discontinuity in usage periods, usage purpose, freedom of user and lack of consideration of business policies and service levels for self-optimization (Huebscher & McCann 2008).

Discontinuity in usage, as presented by (Inmon 2005), separates the utilization periods of analytical DWs into two main periods: use and non-use. (Figure 1).

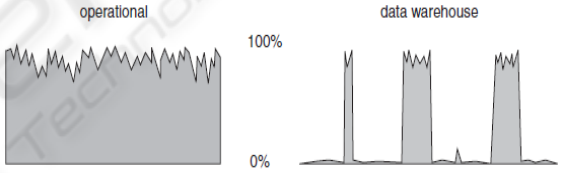

Figure 1: Operational and data warehouse use charge (Inmon 2005).

Due to discontinuity, we are unable to predict the real activity or the resource needs of the DWs. Commonly in enterprises cache allocations are made accordingly to editor recommendations (e.g. allocate the maximum RAM memory possible into cache). In practice, due to the immense sizes of DWs, the allocation is done proportional to the DW size. And, once configured, **cache allocations don't change, though usage conditions change.** This means that cache parameter values don't change over time, from their initial settings.

The usage purpose makes a difference between the moments and purposes of accessing data from the DW. There are two usage purpose intervals. The *day usage*, during the daily 'work hours' (i.e. from 8am to 22pm), provides the users with the data for generating analysis reports. The *night (or batch) usage*,(i.e. from 22pm to 8am), which is 'non-user stressing', during which the data is recalculated and

aggregated with the new incorporated operational data from the day that has just passed. These operations are time consuming and should end before the start of the next operational day to avoid inconsistencies. Therefore, during night all unused system resources should be redirected to this end.

Freedom of the user refers to the ability of any user to create and modify reports at will, based on the data from the DW. Thus, considerations for resource scaling (such as cache allocation) and prediction of the retrieved data must be rethought in the decisional world (Inmon 1995). In their paper (Saharia & Babad 2000), the authors showed how the performances of data warehouse for ad-hoc query response times can be improved by proposing a mechanism for storing into cache the queries (and their results) that are most likely to be queried. Due to the fact that there is a redundancy with the data retrieved from the data warehouses, using cache memories is a common practice to store some of the already required information.

Lack of consideration of service levels for self-optimization is one of the biggest problems with AC adoption today. Usually the focus is on improving technical raw indicators (such as query response times) over service policies (such as query response times scaled with the importance and priority of the DW). In (Vassiliadis et al. 1999), a DW quality model is presented based on the goal-question-metric, thus defining the notion of Quality of Service (QoS). DW 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. They emphasize that data quality is defined as the fraction of performance over expectancy, based on objective quality factors, which are computed and compared to users' expectations. The advantages of a quality-driven model result from the increase of service levels, and along with it, customer satisfaction. One of the disadvantages though is that performance goal metrics are more difficult to define. In (Codd et al. 1993), the authors elaborated certain rules for best practices with OLAP. Some of these are translated by a general rule well known by the experts: that 80% of the OLAP queries must be under a second and it can be interpreted as a performance goal for DWs.

Based on the limits presented, *we have two main objectives* for this paper. *The first* is to show that only improving the technical performances is a common mistake with DSS, and to determine how service improves when QoS is considered as performance indicator over a technical query response time raw indicator (QRT). *The second* objective is to propose a solution for autonomic adoption with DSS, with the help of semantic technologies.

2.2 Architectural Model

First we take a look at the previous work on modeling a DSS and on autonomic adoption. Based on this work, we present our modeling proposition with AC adoption.

The starting point is the AC adoption cube (Figure. 2) presented by (Parshar & Hariri 2007), and developed from the IBM AC specifications (IBM 2001). The cube contains three axes. OX describes the level of autonomic adoption, from manual to fully autonomic management (closed loop without human intervention). OY contains the managed resources (e.g. in our case the elements of the DSS). The description leads to the idea of a hierarchical architecture (starting from OLAP bases as the sub component, and ending with the entire managed system). Last, OZ contains the service flows, both with the proposition of new services and the improvement of existent ones.

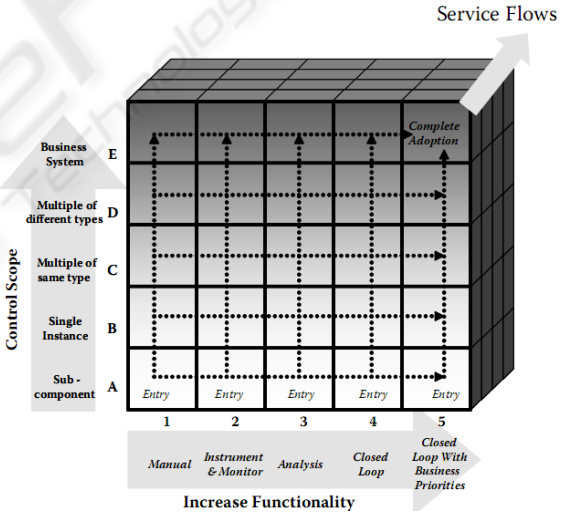

Figure. 2: Autonomic Computing adoption cube (Parshar & Hariri 2007).

Starting from the adoption cube, (Stojanovic et al. 2004) proposed an architectural model for AC adoption for resolving errors related to component availability within a system. They based their approach on the correlation and inference capabilities of the semantic technologies, such as ontologies. By its latest definition, an ontology describes a set of representational primitives (classes, attributes and relationships) used to model a domain of knowledge or discourse (Liu & Özsu

2008). In (Stojanovic et al. 2004), the authors divided the reference model between a resource layer (system architecture), an event layer (error messages) and a rule layer (how to act when faced with various situations). From this approach, in (Nicolicin-Georgescu et al. 2009) the authors presented an AC adoption model for a DSS with the purpose of improving the performances of DWs. They proposed three architectural layers divided into two aspects. The static aspect, which contains the layers: system architecture and parameter/ performance indicators. The dynamic aspect, which contains layer of: advice, best practices and human experience (where the use of web semantics is reinforced).

Our proposition develops the previous approaches, embracing the use of ontology for knowledge formalization. However, we have integrated the approach in a general simplified DSS architecture described in Figure 3. In this architecture, we can distinguish the description of the managed elements and of the management policies via intelligent control loops.

First, there are two types of managed elements: (i) the Physical Server and (ii) the OLAP Base. A physical server contains several OLAP bases, which describe the DW. Between the servers and the bases, a link of inclusion is created to describe which bases belong to which server. Moreover, each managed element has several characteristics and parameters, such as RAM memory, cache memory, average QRT etc.

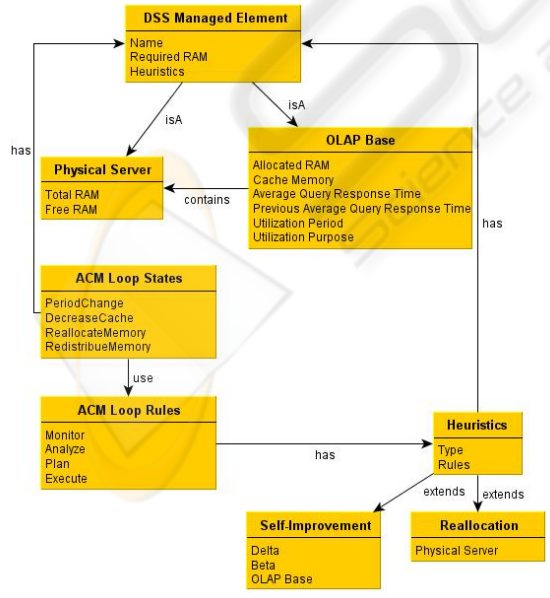

Figure 3: DSS Architectural model.

Second, we treat the problem of shared RAM memory allocation over the OLAP bases, through AC adoption. This leads to the modelling of intelligent autonomic loops over each managed element along with the states and the rules that describe loop behaviour. With the loop description, we have modelled two heuristics, via the Heuristics class: Self-Improvement for the OLAP bases and Relocation for the physical servers.

In our applicative framework, the architectural model is translated under the form of an OWL ontology, with over 150 concepts, 250 axioms and 30 rules. An example of how an OLAP Base is described with the usage of OWL triplets can be seen in Table 1.

Table 1: OWL base example.

Subject	Predicate	Object
?base	rdf:type	Base
?server	rdf:type	PhysicalServer
?server	contains	?base
?base	hasAvgQRT	'1,2'^^xsd;double
?base	hasCacheValue	'500'^^xsd:double
?base	hasUtilPurpose	'Day'^^xsd:string

The intelligent loop and AC adoption has been modelled by the use of a State class, which contains a list of states corresponding to the loop phases. Its elements are exemplified in Figure 4, containing a screen capture from the Protégé open source knowledge modelling framework (Stanford Center for Biomedical Research 2010) which we user for building ontologies. Despite the fact that it scales poorly with big ontologies (not our case), it has the advantages of simplicity, integrations with reasoners and a large supporting community.

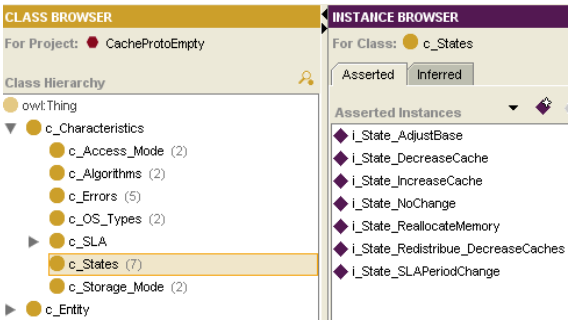

Figure 4: The States used for the intelligent loop.

3 THE MAPE-K LOOP

In the DSS model proposition we have used the term of 'intelligent control loop' to describe how AC adoption is implemented. In this section we detail

this specific aspect, along with the two proposed loop heuristics: self-improvement and reallocation.

Returning to IBM's blueprint, the AC adoption model indicates a self-X factor for reaching autonomy: self-configuration, self-optimization, self-healing and self-protection, implemented by an ACM. The ACM describes an intelligent control loop in four phases: Monitor, Analyze, Plan and Execute. The loop revolves around a Knowledge Base that provides the information needed for its execution. This is why the loop is also known under the name of the MAPE-K loop.

One drawback, so far, with AC adoption is the lack of standardization. In (Vassev & Hinchey 2009) the authors present the advantages and disadvantages of AC adoption, and they propose a software description model language for systems implementing AC. The survey of (Huebscher & McCann 2008) presents, among other things, this particular aspect and underlines the fact that autonomic system are less able to deal with discrete behaviours. They show the drawbacks of implementing reinforcement-learning or feedback self-improvement techniques with the ACM loop. The time to converge increases (with the learning curve) and the implementations are not easily scalable (with a higher number of states). Also, the authors highlight that there is a lack of Service Level Agreements (SLA) implementations with ACM loops, as most of the time the followed performance indicators are technical (e.g. QRT) and not service oriented (e.g. QoS). In (Ganek & Corbi 2003), an SLA is defined as a contract between a customer or consumer and a provider of an IT service; it specifies the levels of availability, serviceability, performance (and tracking/reporting), problem management, security, operation, or other attributes of the service, often established via negotiation.

In our approach, the MAPE-K loop adoption is based on the proposed architectural model and on the limits presented earlier. Figure 5 describes the autonomic manager implementation over the managed elements, under a form of a tree. There are two particular aspects with this adoption.

First, each of the managed element has its own independent ACM. The ACM that has the same behaviour for each tree level (same behaviour for all OLAP bases, same behaviour form all Physical Servers). In our case the managers will be responsible for the adoption of the two heuristics for self-imprvement and RAM reallocation . The ACMs presented in the figure are part of the lowest levels of IBMs AC adoption architecture.

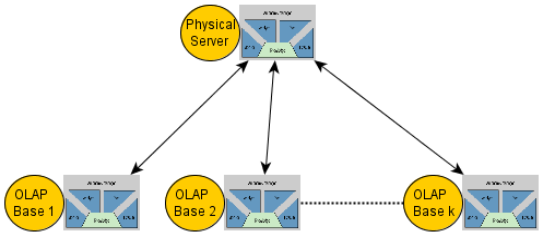

Figure 5: MAPE-K loop adoption over the DSS model.

Second, the communication between managers from different levels is done via the common upper level. If the manager of Base 1 wants to communicate with the manager of Base k it must communicate first with the manager of the common physical server. This choice is based on the hierarchical characteristics of a DSS as indicators aggregate over upper levels, such that expression of a problem to a higher level (e.g. Physical Server) can then be divided and tracked down to lowers levels (e.g.. OLAP Base).

From the presented model we thus have *two* ACM implementations, one for the Physical Server and one for the OLAP Bases. Each of the ACM implementations corresponds in turn to one the two heuristics, described further on.

3.1 Self Improvement Heuristic

This heuristic is implemented by OLAP Base ACM, and has the objective of minimizing the cache allocations while maintaining the query response times at acceptable levels. The idea behind this heuristic is that we can afford to lose a bit in performances if it spears a chunk of free memory (and further gain of more performance after reallocation). With each loop, the caches decrease and, in function of the ratio between the current performance level and the previous levels, the new sizes are either accepted or rejected. For this heuristic we define two variables: a cache modification rate $\boldsymbol{\Delta}$ and a threshold limit $\boldsymbol{\beta}$. The $\boldsymbol{\Delta}$ represents the rate (in percentage) at which the caches modify with each loop passage and the threshold $\boldsymbol{\beta}$ represents the maximum accepted impact that a cache change can have on the performances. Over this limit a $\boldsymbol{\Delta}$ change in cache is no longer accepted.

The self-improvement heuristic is described in Figure 6. It is composed of seven successive steps over the ACM loop phases.

Step 1 (Monitor). At day *n* (until the heuristic started working) set the cache values (CV_n) accordingly the previous ones. CV_{max} represents the maximum theoretical size of the caches (equal to the size of the base): $CV_n = CV_{n-1}*(1 - CV_{n-1} / CV_{max}*\Delta)$
Step 2 (Monitor). Monitor of the impact of the new cache on the response time throughout the next day. Compute the average query response time from this day: $AvgQRT_n$.
Step 3 (Analyze). Compute the average QRT for **all** of the *n-1* previous days (not only the last day): $AvgQRT_{[1,n-1]}$
Step 4 (Analyze). If the levels of performance are accepted: $AvgQRT_n / AvgQRT_{[1,n-1]} < \beta$ go to step 6. Else go to Step 7.
Step 5 (Analyze). If the minimum cache thresholds are not overtaken: $CV_n > CV_{max}$ go to step 6. Else go to Step 7.
Step 6 (Plan&Execute). Accept the new cache levels and restart on the next day. Go to step 1 with $n = n + 1$.
Step 7 (Plan&Execute). Stop and wait for the algorithm to be retriggered (we'll see further on by the cache reallocation heuristic).

Figure 6: Self-improvement heuristic over the ACM loop phases.

An ontology rules description example of the analysis phase, Step 4 and 5, is shown below:

```
RULE 1: (?base rdf:type Base) (?base isChildOf ?srv)
    noValue(?srv, hasState, i_State_ReallocateMemory)
    (?base hasPrevAvgResponseTime ?prev_avgt) (?base hasAvgResponseTime ?avgrt)
    sum(?s, ?prev_avgrt, ?avgrt) quotient(?ratio, ?avgrt, ?s)
    (?hrs rdf:type Heuristics) (?hrs hasBeta ?beta) le(?ratio, ?beta) =>
    (?base hasState i_State_DecreaseCache)
RULE 2: (?base hasState i_State_DecreaseCache) (?base hasCacheValue ?cv)
    (?base hasMinCacheValue ?min_cv) (?base hasMaxCacheValue ?max_cv)
    (?hrs rdf:type Heuristics) (?hrs hasDelta ?delta) difference(?max_cv, ?cv, ?d)
    quotient(?delta, ?d, ?qcv) difference(?cv, ?qcv, ?new_cv) le(?new_cv, ?min_cv) =>
    drop(0)
```

RULE 1 tests for an OLAP base, whether the current day corresponds or not to a reallocation action (no reallocation over the server). If this is not the case, it passes the base into a decrease cache state allowing it to continue the analysis. Next, ***RULE 2*** follows and tests if a base is in the decrease state (so self-improvement heuristic can be applied) and if the decrease is possible (the cache value remains over the minimum cache threshold). If the minimum threshold is reached then the DecreaseCache state is deleted from the base and the algorithm stops.

The efficiency of this heuristic is measured in the time (number of days – loop passages) needed to stabilize itself (stop the algorithm), thus its performance is based on the time to converge. By successive repetitions at a certain point Step 7 is reached and the algorithm stops.

3.2 The Reallocation Heuristic

This heuristic is implemented by the Physical Server managed element (thus the superior tree level). Its objective is to reallocate periodically the freed memory from the self-improvement heuristic, towards the non performing bases, so that the average of QRT ratios is improved globally on the server. A base is considered non-performing for the server, if its average QRT is greater than the average QRT of the server (averages of the base averages QRT). Otherwise, the base is considered performing. The idea here is that in parallel with the first heuristic, it allows by the small sacrifice of certain bases to gain important performance on others.

The reallocation heuristic is triggered either periodically (i.e. each *x* days) or whenever there is a change in the utilization periods. The reallocation heuristic is described in Figure 7. It is composed of four successive steps over the ACM loop phases.

Step 1 (Monitor). At day *n* ($n \bmod x = 0$ or utilization period change). Monitor the average QRT for each of the server's bases. Monitor also the free available memory: FAM.
Step 2 (Monitor). Compute the average server QRT: $AvgQRT_{srv}$.
Step 3 (Monitor). If base x is non performing, $AvgQRT_{base\ x} > AvgQRT_{srv}$ then reallocate a part of the free memory. Otherwise go to Step 4. If we consider no_{nob} the total number of non performant bases, then we have: $CV_{n+1} = CV_n + FAM / no_{nob}$
Step 4 (Plan&Execute). Base x is performing, don't change its cache allocation: $CV_{n+1} = CV_n$

Figure 7: Reallocation heuristic over the ACM loop phases.

Again, an example of implementation with ontology rules is given below for Step 3:

```
RULE 1: (?base rdf:type Base) (?base isChildOf ?srv)
    (?srv, hasState, i_State_ReallocateMemory) (?base hasAvgResponseTime ?avgrt)
    (?srv hasavgResponseTime ?srv_avgrt) greaterThan(?b_avgrt, ?srv_avgrt) =>
    (?base hasState i_AdjustBase)
RULE 2: (?base rdf:type Base) (?base isChildOf ?srv)
    (?base hasState i_AdjustBase) (?srv hasFreeAvailableMemory ?srv_fam)
    (?srv hasNumberOfReallocations ?nr) quotient(?srv_fam, ?nr, ?rl_mem)
    (?base hasCacheValue ?cv) sum(?cv, ?rl_mem, ?new_cv) =>
    drop(2) ( ?base hasCacheValue ?new_cv)
```

RULE 1 adds an adjustment state to the base if the server does a memory reallocation (is in the reallocate memory state). ***RULE 2*** looks at the amount of free available memory that the server can redistribute and the number of non performing bases. It divides equally the available memory, thus the non-performing bases gain more cache memory.

4 EXPERIMENTAL RESULTS

We present further in this section the conducted experiments, which reflect the differences between our approach and the current existing situations throughout enterprises, when faced with the shared resource reallocation issue. First we describe the test protocol and then we present the obtained results.

4.1 Experimental Protocol

With our tests we wanted to prove the first benefits of this approach. We focused on two specific tests that isolate the DWs from any other factors than the ones presented here. On a physical server machine with 1GB of RAM, we have taken six Oracle Hyperion Essbase (Oracle 2010) bases, each of them with a size of 640MB. This means a requirement of

3.8GB of RAM memory in contrast to the only 1GB of RAM available. We considered the bases to be identical in size and data, to isolate our experiments. Yet the bases differ with their utilization periods, which are different for each of the 6 bases (over a period of one month). So we have days during which all the 6 bases are used, and others in which only one is. Next we have taken a pool of 10 queries, which are run on each base every day, in order to simulate the activity, thus having the same activity on each base every day. An example of such query, as an Essbase report, can be seen below. It demands the data of all the possible scenarios for the first quarter of each year, for a product from a specified market.

```
<Sym <Supshare
<Column (Scenario, Year)
        <CHILDREN Scenario  <IDESCENDANTS Qtr1
<Row (Product, Market)
        <IDESCENDANTS ProdB   <IDESCENDANTS MarketA
```

The average response time for such a query, if the information is not stored into caches is 10 seconds. With the data cached, it goes down to 2 seconds.

The test protocol is described in Figure 8 as follows

Step 1 (Monitor). Simulate the daily activity over the application. Run the 10 queries over each of the 6 base, in a sequential way. The query response time for each query/base are recorded in the log files on the server.
Step 2 (Monitor). From the log files, build SQL tables and views to obtain the QRT and average QRT over each base.
Step 3 (Analyze). Load in the ontology model the average values that interest the heuristic.
MAPEK loop for the heuristics Step 4. Run the heuristics over the OWL ontology and propose new cache values for the bases.
Step 5 (Execute). Change the values of caches for the bases via VB and MAXL scripts for Essbase connection.
Step 6 (Execute). Once the new values changed a cycle is over and we can go over to the next day. Go to **Step 1**

Figure 8: Test protocol steps over the ACM loop phases.

Over this protocol, we have carried out two different tests. We wanted to emphasize the difference between the evolutions of the levels of service when taking in consideration the technical performance indicator only (average QRT) and when integrating the objective QRT (QoS).

The first test, technical, ran the heuristics while taking the average QRT as performance indicator for the heuristics.

The second test, service, ran under the same conditions, but this time the performance indicator used in the heuristics was the goal performance indicator, or a service QRT. We propose a definition of the level of satisfaction, a QoS indicator, based on the objective QRT expressed by Codd for the OLAP bases. Thus we consider the objective performance QRT at 1 second, and compute the level of user satisfaction in rapport with this level. The further an average QRT is from 1 second, the lower the satisfaction and the level of service. Therefore we proposed a formalization of the levels of service and define the $QoS = AvgQRT_{target} / AvgQRT_{current}$. (with $AvgQRT_{target} = 1$).

4.2 Results

Using the two test scenario above, we present in Figure 9 the results obtained. The graph is done for a period of 21 days (one working month). The vertical lines indicate a utilization period change for any of the OLAP bases. After each such change, the performances greatly increase from the reallocation heuristic, and then slightly decrease from the self-improvement one.

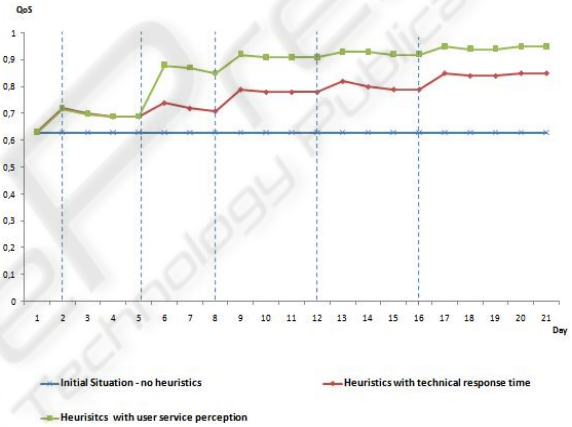

Figure 9: QoS comparison between a common constant configuration, heuristics over technical indicators and heuristics over service levels.

We analyze each of the three situations from Figure 9.

In the first one, a normal situation that happens today in enterprises, where configurations don't change and usage periods are not taken into consideration, thus the QoS remains the same.

In the second situation we see the results with our proposal where usage periods are taken into consideration. Here we can see an improvement over the QoS, as taking into consideration usage periods renders the bases that are in activity more important. This allows them to have more resources and thus increased QoS. Still the heuristics performance indicator remains the technical QRT.

The third situation shows the results with our approach where we: (a) take into consideration usage periods, (b) modify the query response times according to these usage periods, and (c) integrate the heuristics performance indicator as the user

perceived service QRT. In this case we can see even a higher improvement in the QoS, starting from the 7th day. Up to this point, as there is no reallocation, the technical and service QRT is identical, thus the QoS is identical. The QoS starts improving from that day, when the reallocation is made in function of the service QRT.

5 CONCLUSIONS

In this paper we have presented an approach to managing decision support systems via data warehouse cache allocations by using autonomic computing and semantic web technologies. By considering the specifications and characteristics of DSS, we showed how AC adoption can be enabled with DW resource allocation. We have presented two heuristics for AC adoption and have based our approach on semantic web technologies by using ontologies for DSS system modeling and ontology based rules for heuristics and ACM loop implementation.

With the results we have shown that taking into consideration QoS over raw technical indicators is a must and one important step forward with AC adoptions over DSS.

In the next future we intend to extend the notions of SLA in order to study further how performance and QoS are influenced when using more SLA considerations. We also want to extend the influence factors over the activity of the DWs, by extending the perimeter of resource allocation (i.e. CPU charge, disk usage etc.) and the perimeter of performance measure such as calculation times.

REFERENCES

Codd, E. F., Codd, S. & Salley, C. (1993), 'Providing olap to user-analysts: An it mandate'.

Frolick, M. N. & Lindsey, K. (2003), 'Critical factors for data warehouse failure', *Business Intelligence Journal* 8(3), 48–54.

Ganek, A. G. & Corbi, T. A. (2003), 'The dawning of the autonomic computing era', *IBM Systems Journal* 43(1), 5–18.

Huebscher, M. & McCann, J. (2008), 'A survey on autonomic computing – degrees, models and applications', *ACM Computing Surveys* 40(3), 1–28.

IBM (2001), *An architectural blueprint for autonomic computing*, IBM Corporation.

Inmon, W. H. (1995), *Tech topic: what is a data warehouse?*, Prism solutions, Volume 1.

Inmon, W. H. (2005), *Building the data warehouse, fourth edition*, Wiley Publishing.

Lightstone, S. S., Lohman, G. & Zilio, D. (2002), 'Toward autonomic computing with db2 universal database', *ACM SIGMOD Record* 31(3), 55–61.

Liu, L. & Özsu, M. T. (2008), *Encyclopedia of Database Systems*, Springer-Verlag. http://tomgruber.org/writing/ontology-definition-2007.htm

Maedche, A., Motik, B., Stojanovic, L., Studer, R. & Volz, R. (2003), 'Ontologies for enterprise knowledge management', *IEEE Intelligent Systems* 18(2), 26–33.

Markl, V., Lohman, G. M. & Raman, V. (2003), 'Leo : An autonomic optimizer for db2', *IBM Systems Journal* 42(1), 98–106.

Mateen, A., Raza, B. & Hussain, T. (2008), Autonomic computing in sql server, *in* 'Proceedings of the 7th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2008', 113–118.

Nicolicin-Georgescu, V., Benatier, V., Lehn, R. & Briand, H. (2009), An ontology-based autonomic system for improving data warehouse performances, *in* 'Knowledge-Based and Intelligent Information and Engineering Systems, 13th International Conference, KES2009', 261–268.

Oracle (2010), 'Oracle Hyperion Essbase'. http://www.oracle.com/technology/products/bi/essbase/index.html

Parshar, M. & Hariri, S. (2007), *Autonomic Computing: Concepts, Infrastructure and Applications*, CRC Press, Taylor & Francis Group.

Saharia, A. N. & Babad, Y. M. (2000), 'Enhancing data warehouse performance through query caching', *The DATA BASE Advances in Informatics Systems* 31(2), 43–63.

Sirin, E., Grau, B., Grau, B. C., Kalyanpur, A. & Katz, Y. (2007), 'Pellet: A practical owl-dl reasoner', *Web Semantics: Science, Services and Agents on the World Wide Web* 5(2), 51–53.

Stanford Center for Biomedical Informatics Research (2010), http://protege.stanford.edu/

Stojanovic, L., Schneider, J. M., Maedche, A. D., Libischer, S., Studer, R., Lumpp, T., Abecker, A., Breiter, G. & Dinger, J. (2004), 'The role of ontologies in autonomic computing systems', *IBM Systems Journal* 43(3), 598–616.

Vassev, E. & Hinchey, M. (2009), 'Assl: A software engineering approach to autonomic computing', *Computer* 42(6), 90–93.

Vassiliadis, P., Bouzeghoub, M. & Quix, C. (1999), Towards quality-oriented data warehouse usage and evolution, *in* 'Proceedings of the 11th International Conference on Advanced Information Systems Engineering, CAISE 99', 164–179.