System Models for Goal-Driven Self-management in Autonomic Databases

Marc Holze and Norbert Ritter

University of Hamburg, Department of Informatics,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{holze,ritter}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/

Abstract. Self-managing databases intend to reduce the total cost of ownership for a DBS by automatically adapting the DBS configuration to evolving workloads and environments. However, existing techniques strictly focus on automating one particular administration task, and therefore cause problems like overreaction and interference. To prevent these problems, the self-management logic requires knowledge about the system-wide effects of reconfiguration actions. In this paper we therefore describe an approach for creating a DBS system model, which serves as a knowledge base for DBS self-management solutions. We analyse which information is required in the system model to support the prediction of the overall DBS behaviour under different configurations, workloads, and DBS states. As creating a complete quantitative description of existing DBMS in a system model is a difficult task, we propose a modelling approach which supports the evolutionary refinement of models. We also show how the system model can be used to predict whether or not business goal definitions like the response time will be met.

1 Introduction

For several years, researchers and DBMS vendors have developed self-management functions for particular DBMS components or administration tasks. Thus, commercial DBMS today typically provide an index advisor, autonomic memory management, and automated maintenance functions. These self-management approaches usually follow the feedback control loop (FCL) pattern, where a *controller* continuously monitors a resource via *sensors*, analyses the observed information, and immediately performs necessary reconfigurations via *effectors*.

Already at an early stage of autonomic DBS technology development it has been noticed [1] that adding FCLs to a DBS entails severe problems. These problems are primarily caused by the fact that the FCLs lack knowledge about the components they manage: First, they cannot predict the performance effect of planned reconfigurations on the managed resource. Autonomic memory management, for example, "guesses" the required memory adaptations in small steps of fixed size. This trial-and-error bears the risks of *overreaction*, because DBS reconfigurations may require a long time before the reconfigurations take effect.

J.D. Velásquez et al. (Eds.): KES 2009, Part II, LNAI 5712, pp. 82–90, 2009.

Second, the FCLs lack knowledge about the system-wide effects of their reconfigurations. For example, an autonomic connection management may increase the number of DB agents to improve the overall system throughput. However, this decision may also affect the performance of other components, as it reduces the memory available for the system buffer. The lack of knowledge about side-effects causes *interference* between the decisions of the FCLs. Third, the relationship between the performance of the managed component and system-wide goals (e.g. response time, availability) is unknown to the FCLs (*goal-independency*). Knowledge about the relationship to system goals would allow the FCLs to restrict expensive reconfiguration analyses to situations when they are actually required.

In this paper we present an approach towards system-wide, goal-driven DBS self-management. In particular, we identify the knowledge that is required to prevent the problems of overreaction, interference and goal-independency. We propose the usage of a system model, which serves as a centralized knowledge base for the self-management logic. By choosing the graphical modelling language SysML for the system model, our approach is designed to be used for existing DBMS. We show how knowledge about both the structure and the behaviour of the DBMS can be expressed in this model, and how the system model can be extended by objective functions that can be evaluated against goal definitions.

The paper is structured as follows: Section 2 discusses the necessary system model contents and the relevant goal types. In Section 3 we describe how a system model can be created for existing DBMS, using IBM DB2 as an example. The objective functions for the goal-driven optimization is subject of Section 4. We discuss related work in Section 5 before we conclude with an outlook in Section 6.

2 Goal-Driven Self-Management

In order to reduce the operation and maintenance costs, DBS should automatically consider the users' performance requirements (goals) during query processing, e.g. by treating queries with high priority preferentially. However, as no existing DBMS provides this functionality, our objective is to create an external self-management logic, which can be applied to existing DBMSs. Hence, meeting the goals must be accomplished by using the existing configuration options of a DBMS only. For this purpose, it is essential for the self-management logic to maintain a system-wide view on the DBS. As shown in Fig. 1, the *workload* and the *state* of the DBS must be continuously monitored by a *self-management logic*. The workload provides information on how a DBS is used in its particular environment (e.g. CPU usage, SQL trace). The state information refers to internal characteristics like the average response time or the accuracy of optimizer statistics. The self-management logic must compare the current state of the DBS to the *goals* defined by a DBA, and start a reconfigurations analysis when there is a risk of missing the goals. For the decision on which reconfigurations will meet the goals under the current workload, the self-management logic needs detailed knowledge about the DBS. We refer to this knowledge as the *system model*.

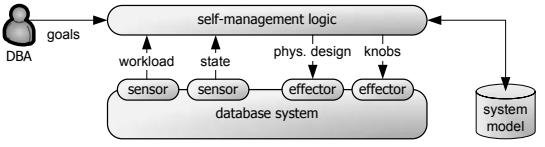

Fig. 1. DBS Self-Management Overview

From the knowledge stored in the system model, it must be possible to decide how the computing resources must be shared amongst the DBMS components. In addition, the self-management logic must be able to assess configuration alternatives (access paths, number of bufferpools, tablespace design, ...) in a specific environment. So the system model must comprise two main parts: a *Hardware Model* describing the available computing resources, and a *DBMS Model* describing the DBMS components and their behaviour. It is the task of self-management logic to evaluate the knowledge stored in the system model in order to decide which reconfigurations are necessary to meet the goals. Hence, the system model must also comprise *goal functions*. These functions must quantitatively describe the values expected for the goals response time, throughput, CPU/disk usage, availability, and operation costs, depending on the DBS configuration.

3 System Model

In the following, we describe our approach for storing the knowledge for DBS self-management in a system model. Section 3.1 first identifies the necessary model contents, before Section 3.2 describes the selected modelling technique.

3.1 Model Contents

As described in Section 2, the system model stores knowledge about the structure of the DBMS and the available hardware. The DBMS model is hierarchically structured (see Fig. 2), where every level refines the component structure. For every component the sensors, effectors and constraints must be defined:

As the *sensor* data must be read and interpreted by the self-management logic, the DBMS model must describe how the sensor value can be retrieved (*availability*, e.g. from system catalogue). Furthermore, the meaning of the sensor information (*semantics*, e.g. counter, or high water mark), and its *type* (workload or state) have to be defined. For *effectors* it is necessary to describe their *type* (e.g. configuration parameter, physical design, maintenance function) and whether they can be manipulated online (*changeability*). The effector values of DBMS components in many cases may not be changed arbitrarily, but are subject to *constraints*. For example, there usually are rules for the allowed values (*domains*). In addition, configuration parameters of one component may depend on configuration values of another component (*dependencies*). The most important constraints for the automatic deduction of reconfiguration actions is the description of the expected *behaviour* of the component. It must be described in terms

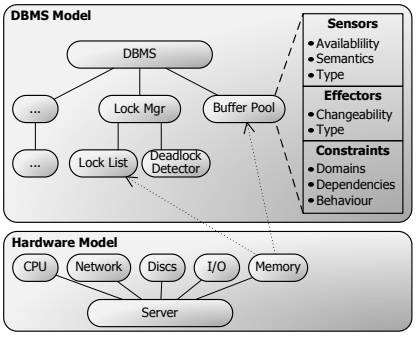

Fig. 2. DBMS- and Hardware-Model

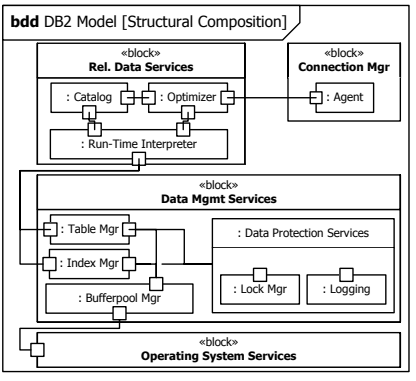

Fig. 3. DB2 Structure Example

of a mathematical model of the component, which quantifies the performance of the component depending on its sensor and effector values. Only then it is possible for the self-management logic to predict the effects of reconfigurations.

It is important to note that the description of the logical and physical design of a DBS is *not* part of the DBMS model. This information is instead available from the system catalogue as sensor information, and it can be adapted via effectors in order to influence the performance of the DBMS components.

The hardware information in the system catalogue usually does not provide information on the performance characteristics and costs of using a particular piece of hardware. However, this information will be required by the self-management logic in order to minimize computation cost, while meeting the performance goals. Hence, a separate hardware model must be maintained. This model will also allow the explicit representation of resource competitions between DBMS components (illustrated as dotted arrows in Fig. 2).

3.2 DBMS System Modelling

Due to the complexity of today's commercial DBMSs, a complete and exact mathematical model of the system behaviour is not feasible. But as noted by Weikum et al. in [1], even an approximate, coarse-grained model would be a step ahead. Hence, our approach towards a system model is to create a model outside of the self-management logic. In contrast to implementing the knowledge about the system structure and behaviour directly in the logic, placing it in a separate model has the advantage that it can be easily extended, refined and adapted. So a very coarse-grained model can be created in a first step, containing the most important components, a subset of their actual sensors and effectors, and an approximated quantification of their behaviour. After this model has proven to sufficiently well predict the system behaviour, it should be refined incrementally.

For the definition of a system model we have chosen the SysML modelling language [2], because it allows the representation of all the required model contents identified in Section 3.1. SysML has been designed to support the specification,

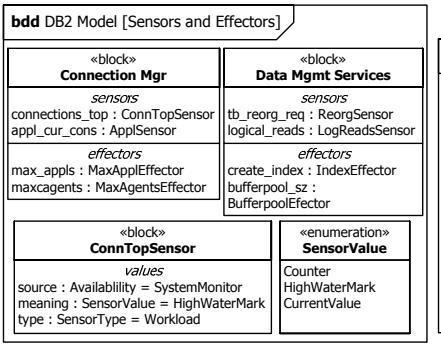

Fig. 4. Sensors/Effectors Example

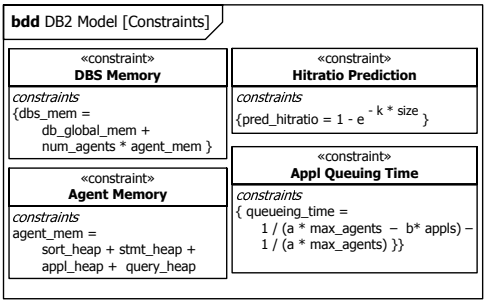

Fig. 5. DB2 Constraints Example

analysis, design, and validation of a broad range of systems and systems-of-systems, including hardware, software, information, and processes aspects [3]. For the definition of the structure of the system as well as the constraints on effector values we use *block definition diagrams* and *internal block diagrams*. For the quantification of the system behaviour we employ *parametric diagrams*.

We illustrate the usage of SysML for the definition of DBS system models by describing an exemplary model for IBM DB2. A coarse-grained structural model of DB2 is given in terms of a SysML block definition diagram in Fig. 3. It shows the hierarchical structure of the main DB2 components: The *Connection Manager* handles the connections to client applications by assigning *Agents* to them, which execute the SQL statements in the DBS for the client. The agent passes the statements to the *Relational Data Services*, where execution plans are created for them using the *Optimizer* and *System Catalog*. These plans are executed in the *Run-Time Interpreter*, where they use the *Table Manager* and *Index Manager* components of *Data Management Services* to retrieve the data. The Table Manager and Index Manager request the necessary pages from the *Bufferpool Manager*, which accesses the files on disk via the *Operating System Services*. Isolation and Durability are ensured by the *Data Protection Services*.

The block diagrams in Fig. 3 are well suited to depict the structural composition of a DBMS. For the definition of the components' sensors and effectors we use an additional block definition diagram. Examples for sensor and effector definitions of DBMS components are given in Fig. 4. The Connection Manager, for example, has a sensor *connections_top* for the highest observed number of client connections. Every sensor and effector has to be represented by a separate block in the model, as specific characteristics must be stored (e.g. *ConnTopSensor*).

In addition to the structural information about DBMS components, the DBMS model has to describe the effectors' domain constraints, dependencies, and the component behaviours. For these purposes the SysML ConstraintBlock element is used. The left part of Fig. 5 shows an example for two constraints *DBS Memory* and *Agent Memory*, which define the dependencies between the sizes of different memory areas. It shows that the memory available in the DBS global memory (e.g. bufferpools) depends on the number of agents. The right part of

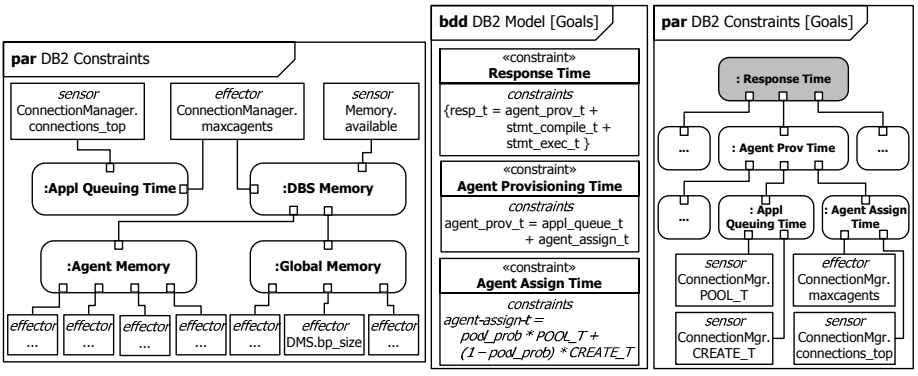

Fig. 6. Constraints Instances

Fig. 7. Objective Functions Example

Fig. 5 depicts two examples for the quantification of the behaviour of DBMS components: The *Hitratio Prediction* quantifies the expected hitratio for bufferpools depending on their size. The average time applications have the wait for an agent is approximated by *Application Queuing Time*.

To evaluate the knowledge about dependencies and component behaviour in a DBMS, the parameters of the constraints have to be mapped to the corresponding sensors and effectors. This task is the subject of the SysML parametric diagram shown in Fig. 6. A self-management logic evaluating this model would find that the expected application queuing time could be reduced by increasing the number of agents (*maxcagents*). However, as this effector is also subject to the *DBS Memory* constraint, the logic would have to find a trade-off.

4 Objective Functions

The system model described in Section 3 serves as a knowledge base for the self-management logic. It predicts the behaviour of the system under different DBS configurations. However, in order to decide whether or not the goals defined by a DBA will be met, the description of the system behaviour must be related to the goal definitions. Hence, we extend the system model by *objective functions*. These are modelled as additional constraints and represent the business goals response time, throughput, resource usage, availability and operation cost. Each of these objective functions must quantitatively describe how its value depends on the DBS configuration. The block definition diagram in Fig. 7 illustrates an example for a *Response Time* constraint, which is quantified as the sum of the waiting time for an agent (*agent_prov_t*), the time required to determine an execution plan (*stmt_compile_t*) and the time for executing the plan (*stmt_exec_t*). In order to be able to actually predict the goal values, these objective functions must be refined until they depend only on sensor and effector values. For this purpose it is possible to either re-use the component behaviour descriptions, or add new

constraints. An example for objective function refinement is described for the response time goal in Fig. 7: In the block definition diagram we first define two additional constraints *Agent Provisioning Time* and *Agent Assign Time* (time for assigning an agent depending on whether or not an agent is available in the pool). These constraints are then used in the parametric diagram in a hierarchy of constraint instances, which define how the response time depends on a set of sensor and effector values. As shown in this example, the objective function constitutes the top of the constraint hierarchy, whereas the leaf nodes must be sensors and effectors described in the system model.

With the objective functions being defined in the system model, the task of self-management is to find an optimal set of effector values such that the function values are minimized (e.g. response time, operation cost) or maximized (throughput, availability). So the challenge of self-management is to find an optimal set of effector values for several, possibly opposing, objective functions. Generally speaking, the self-management has to determine an optimal configuration $\boldsymbol{x}$ for an objective vector $F(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots f_k(\boldsymbol{x}))$, where $f_1(\boldsymbol{x})$ to $f_k(\boldsymbol{x})$ define objective functions. In addition, it has to take into account the constraints on effector values. Finding a solution to this type of problem is the subject of multi-objective optimization (e.g. [4]).

5 Related Work

Research in the area of self-managing database systems focuses on the automation of individual administration tasks like memory management ([5]) or on-line index selection [6]. These autonomic functions do not consider relationships to other autonomic managers, side-effects, or high-level goals. Recently, also works on meeting response-time goals for multiple service classes in DBS have been published ([7], [8]). These approaches strictly focus on admission control for queries, and do not consider the adaptation of the DBS configuration.

Models with quantitative descriptions of the managed resource behaviour by now have only been used for bufferpool management in DBS. These models are used to predict the bufferpool hit ratio ([9], [10]) depending on the bufferpool size, thus allowing the definition of target service times for individual page requests. Brown et al. extend their bufferpool hit ratio prediction model in [11] by making the simplified assumption that there is a linear dependency between the hit ratio and the overall DBS response time. Unlike our approach, all quantitative models for DBS strictly focus on the bufferpool component, and do take into account the influence of the other DBS components and configuration parameters on the overall DBS perfomance.

The Common Information Model [12] provides standardized management of IT systems, independent of the manufacturer and technology. Among others, CIM defines an abstract model for database systems. However, the CIM database system model only describes general information about the DBS, like the instance name, version, the responsible DBA, and the current values of

configuration parameters. The internal structure of the DBMS and a quantitative description of the system behaviour are not part of the model.

The IBM Autonomic Computing Toolkit (ACT) [13] stores information about the resource managed by an autonomic manager in a resource model. This resource model uses CIM classes to define the properties of resources, and stores additional information like check cycles, thresholds, and dependencies. However, reconfigurations cannot be automatically derived from the model, but a decision tree script must be provided, which implements this knowledge.

In order to realize self-management of complex IT infrastructures, rule-based frameworks like Accord [14] and iManage [15] have been developed. Like the ACT, these frameworks require the system administrator to define a set of actions and the conditions under which they are fired. The same approach is taken in policy management frameworks like [16]. So the decision about which reconfigurations have to be performed in order to meet business goals is not derived from a quantitative behaviour description. To overcome this limitation the Accord framework has been extended with a Limited-Look-Ahead-Controller [17]. However, the controller is limited to the optimization of a single objective function, and does not allow the creation and refinement of a system model.

In the area of web service composition there has been research on the usage of multi-objective optimization in order to meet SLAs, e.g. in [18] and [19]. Depending on the QoS requirements, multiple concrete web services are composed to realize an abstract business process. Compared to databases, the objective functions for web service composition are rather simple and the configuration alternatives are limited. However, the results show that using multi-objective optimization for meeting business goals is a feasible approach, and therefore encourage our research in applying these techniques to databases.

6 Conclusions

Currently the knowledge about the sensors and effectors in a DBS, the rules that apply to their values, and their expected effect on the system behaviour is either documented in manuals or the experience of the DBA. Representing this knowledge in a system model allows the creation of a system-wide self-management logic, which can consider the dependencies between reconfiguration actions. In addition, the quantitative description of the system behaviour can be used to ensure that business goals are met. As creating an exact quantitative model of today's complex DBMSs is a difficult task, we have proposed a graphical modelling approach, which allows the step-wise refinement of a coarse grained system model. In the future we are going to realize various coarse-grained system models for different DBMSs and evaluate the accuracy of their behaviour predictions. By comparing the different system models, we are going to identify similarities and common concepts, which will allow us the proposal of a domain-specific modelling language for DBS system models.

References

1. Weikum, G., et al.: Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In: Bernstein, P.A., et al. (eds.) Proc. of the 28th Intl. Conf. on Very Large Data Bases, pp. 20–31. Morgan Kaufmann, San Francisco (2002)
2. Weilkiens, T.: Systems Engineering with SysML/UML, 1st edn. Morgan Kaufmann, San Francisco (2008)
3. Object Management Group: Systems Modeling Language. 1.1 edn. (2008)
4. Coello, C., et al.: Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edn. Springer, Heidelberg (2007)
5. Storm, A.J., et al.: Adaptive Self-Tuning Memory in DB2. In: Dayal, U., et al. (eds.) Proc. of the 32nd Intl. Conf. on Very Large Data Bases, pp. 1081–1092. ACM Press, New York (2006)
6. Bruno, N., Chaudhuri, S.: An Online Approach to Physical Design Tuning. In: Proc. of the 23rd Intl. Conf. on Data Engineering, pp. 826–835. IEEE Computer Society Press, Los Alamitos (2007)
7. Krompass, S., et al.: Quality of Service-enabled Management of Database Workloads. IEEE Data Eng. Bull. 31(1), 20–27 (2008)
8. Niu, B., et al.: Workload adaptation in autonomic DBMSs. In: Erdogmus, H., et al. (eds.) Proc. of the, Conf. of the Center for Advanced Studies on Collaborative Research, p. 13. IBM Press (2006)
9. Tran, D.N., et al.: A new approach to dynamic self-tuning of database buffers. ACM Transactions on Storage 4(1), 1–25 (2008)
10. Chung, J.Y., et al.: Goal-oriented dynamic buffer pool management for database systems. In: Proc. of the 1st Intl. Conf. on Engineering of Complex Systems, pp. 191–198. IEEE Computer Society Press, Los Alamitos (1995)
11. Brown, K.P., et al.: Goal-Oriented Buffer Management Revisited. In: Jagadish, H.V., Mumick, I.S. (eds.) Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 353–364. ACM Press, New York (1996)
12. Distributed Management Task Force: Common Information Model (CIM) Infrastructure. 2.5.0a edn, Specification (2008)
13. IBM Corporation: A Practical Guide to the IBM Autonomic Computing Toolkit. 1st edn., Redbook (2004)
14. Liu, H., Parashar, M.: Accord: a programming framework for autonomic applications. IEEE Trans. on Systems, Man, and Cybernetics 36(3), 341–352 (2006)
15. Kumar, V., et al.: iManage: Policy-Driven Self-management for Enterprise-Scale Systems. In: Cerqueira, R., Campbell, R.H. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 287–307. Springer, Heidelberg (2007)
16. Bhide, M.: et al.: Policy Framework for Autonomic Data Management. In: Proc. of the 1st Intl. Conf. on Autonomic Computing, pp. 336–337. IEEE CS Press, Los Alamitos (2004)
17. Bhat, V.: et al.: Enabling Self-Managing Applications using Model-based Online Control Strategies. In: Proc. of the 3rd Intl. Conf. on Autonomic Computing, pp. 15–24. IEEE Computer Society Press, Los Alamitos (2006)
18. Wada, H., et al.: Multiobjective Optimization of SLA-aware Service Composition. In: Proc. of the IEEE Congress on Services - Part I, pp. 368–375. IEEE CS Press, Los Alamitos (2008)
19. Chang, W.C., et al.: Optimizing Dynamic Web Service Component Composition by Using Evolutionary Algorithms. In: Skowron, A., et al. (eds.) Proc. of the IEEE/WIC/ACM Intl. Conf. on Web Intelligence, pp. 708–711. IEEE CS Press, Los Alamitos (2005)