REACTIVE AUTONOMIC SYSTEM PERFORMANCE MODELING AND SELF-MONITORING WITH CATEGORY THEORY

Olga Ormandjieva, Heng Kuang and Stan Klasa
Department of Computer Science and Software Engineering, Concordia University
Montreal, Quebec, H3G 1M8, Canada

Keywords: Reactive Autonomic System, Performance Modeling, Self-monitoring, Category Theory, Representational Theory of Measurement, Decision Making.

Abstract: The research presented in this paper was motivated by the need to build performance self-monitoring and a decision-making process into Reactive Autonomic Systems (RAS). In order to achieve RAS compliance in terms of the imposed performance policies, we formalize RAS modeling and performance control in a single framework based on a representational theory of measurement and category theory. Category theory is expressive enough to capture qualitative and quantitative knowledge about heterogeneous RAS requirements and their interrelationships, as well as a decision-making mechanism, in one formal representation, where structure and reasoning are inextricably bound together. Thus, category theory provides a computational mechanism which enables this knowledge to be applied to performance data and RAS information structures in order to arrive at valid conclusions.

1 INTRODUCTION

The main obstacle to further progress in the IT industry is software complexity, since the difficulty of managing massive computing systems goes well beyond the capabilities of IT administrators. Some of that complexity derives from the real-time and reactive nature of software systems. One of the solutions to the emerging complexity problem is autonomic computing, which helps by using technology to manage technology. As a result, low level complexities are hidden from end users or removed altogether (IBM Corporation, 2006) (IBM Tivoli, 2005). With autonomic behavior, real-time reactive systems can increasingly self-manage, and be more adaptive to their environment.

Current formal methods have not adequately addressed the issue of verifying policies on behavior, such as performance requirements, which constitute one of the most important nonfunctional requirements for Reactive Autonomic Systems (RAS).

According to (ISO/IEC 9126-1:2001, 2001), level of performance is "the degree to which the needs are satisfied, represented by a specific set of values for quality characteristics." Performance characteristics can be quantified through measurement procedures which provide measurement methods and functions, as well as a meaningful analysis algorithm for combining measurement data along with decision making criteria.

The research proposed in this paper addresses the following challenges:

- The requirement of the performance-critical characteristics of the RAS for specification and for theoretically valid measurement data;
- The need for performance self-assessment to be regulated by policies which state the constraints on system performance fluctuations at runtime.

In order to achieve RAS compliance with the imposed performance requirements, we formalize RAS and performance modeling in a single framework (RASF) based on a representational theory of measurement and category theory. Category theory is expressive enough to capture qualitative as well as quantitative knowledge about heterogeneous RAS requirements and their interrelationships, and a decision-making

Ormandjieva O., Kuang H. and Klasa S. (2009).
REACTIVE AUTONOMIC SYSTEM PERFORMANCE MODELING AND SELF-MONITORING WITH CATEGORY THEORY .
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 325-330
DOI: 10.5220/0002257003250330

mechanism in one formal representation, where structure and reasoning are inextricably bound together. Furthermore, category theory allows for a formal graphical representation of the syntax and semantics of models which goes beyond existing graphical languages, such as UML, where the semantics of the models is informal or semi-formal.

The rest of this paper is organized as follows: Section 2 surveys related work. RAS modeling is described in section 3. Section 4 introduces performance modeling. RAS and performance models are integrated into an RASF Metamodel in section 5, and further formalized in terms of category theory in section 6. Our conclusions are presented and future work directions outlined in section 7.

2 RELATED WORK

This section gives a brief overview of related work on performance modeling and self-monitoring in autonomic systems.

IBM Research has developed a framework called Policy Management for Autonomic Computing (PMAC) (IBM Tivoli, 2005), which provides a standard model for the definition of policies and an environment for the development of software objects that can hold and evaluate policies.

The paper (Abdelwahed and Kandasamy, 2006) describes a model-based control and optimization framework for designing autonomic systems which continually optimizes their performance by changing workload demands as well as operating conditions. The performance management problem of interest can be considered to be one of sequential optimization under uncertainty, and a look-ahead control approach is used to optimize system behavior forecast over a limited prediction horizon. The basic control concepts are then extended to tackle distributed systems where multiple controllers must interact with one another to ensure the overall performance goals.

The research presented here differs from work previously done in the area in an important way: the RAS components, the measurement procedure, and the performance are modeled as categories within the same formal framework, which makes it possible to formalize the self-monitoring policies, verify both their consistency and their completeness, and consequently build performance self-monitoring into the RAS implementation.

3 RAS MODELING

Systems designed to be reactive and autonomic (RAS) are complex and built from potentially very large numbers of elements which are highly autonomic and reactive, but which are also socially interactive. The formal and comprehensive framework used for modeling and controling performance in RAS, the RASF, is built on a 4-tier layered structure (see Figure 1), which includes Reactive Autonomic Objects (RAO), Reactive Autonomic Components (RAC), Reactive Autonomic Component Groups (RACG), and Reactive Autonomic Systems (RAS).

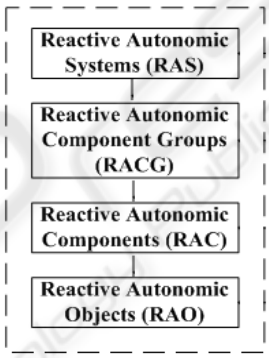

Figure 1: RASF tiers.

The RASF structure is made up of distributed RACG with their asynchronous communication. The RAC is a set of synchronously communicating RAO, where one of the RAO is designated as the leader of the workers. The autonomic behavior, such as self-monitoring or self-analyzing, is implemented by the RAC leaders, group supervisors, and system managers in the RAC, RACG, and RAS tier respectively.

The current trend in autonomic system development is towards the direct or dynamic composition of autonomic components through task workflows. We abstract the behavior of the RAS to a collection of communicating task processes. The workers are mainly responsible for reactive tasks, while the leader works on autonomic tasks such as coordinating self-monitoring at the component level. The assumption here is that each group performs an autonomic task process, and so there is no dependency between the task processes of different groups. RACG are required to react in real time to requests from the system manager for the fulfilment of a task process. The task workflows are scheduled by TACG supervisors, while individual tasks from the specified workflow are assigned to RACs in the group and then optimized, given the resource

constraints of the RAO in real time. The task workflow requires both communication and synchronization of individual tasks to ensure reliable performance by the supervised group, governed by RAS policies.

RAS performance policies impose restrictions on task process communication and synchronization, and so need to be considered as an integral part of the RAS self-management capability. To allow for managing group task process performance, we need to express performance in quantifiable terms by devising an appropriate performance measurement model. We describe next the hierarchical modeling of that performance.

4 MODELING PERFORMANCE

In our approach, performance is modeled as a hierarchical information structure. The performance model proposed in this paper ensures that all standard aspects of quality are considered from both the internal and external points of view. It is decomposed into four qualitative performance characteristics: i) reliability (the capability of the software product to maintain a specified level of performance when used under specified conditions (ISO/IEC 9126-1:2001, 2001); ii) fault tolerance (the capability of the software product to maintain a specified level of performance in cases of software faults (ISO/IEC 9126-1:2001, 2001); iii) efficiency (the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions (ISO/IEC 9126-1:2001, 2001); and iv) performance compliance (the capability of the RAS to adhere to response time, as well as to throughput policies, which are related to task execution and collaboration respectively). Those high-level characteristics are repeatedly refined, and in each of the decompositions the offspring (sub) characteristics can contribute partially or fully towards satisfying the parent. The lowest level corresponds to the quantifiable performance sub characteristics of the RAS tasks computed by applying a measurement method – a logical sequence of operations applied directly to the source, that is, to a task or task process. The result of applying a measurement method is called a base measure. The base measures are then combined by a measurement function to obtain the derived measures required for characterizing the parent (indicator) in accordance with the associated rules for the interpretation of measurement data (ISO/IEC 15939, 2007).

An indicator provides an estimate or evaluation of the utility of the performance characteristic, which is derived from an analysis of the measurement data (values) and with respect to a defined decision criterion. “Utility” in this context means a property in any task process which tends to produce a quality benefit or to prevent disruption (failure behavior or unacceptably low reliability) to an RAC, a RACG, or the whole system.

The combined utility of all indicators serves as a basis for performance self-management decision making on the part of the RACG supervisor.

The self-management decision-making process can be modeled as a set of alternative rules linking the performance utility of a task process to certain actions to outcomes (Roberts, 1979). For example, the input is task process performance utility and the output is the action required to improve the task process performance level in the RAS. If decisions are being made in a situation of certainty, then we choose that action the certain outcome of which maximizes (minimizes) the utility of the task process, depending on the rewards associated with each outcome of an action (Roberts, 1979). The outcome of the action consists of changes to the task process which are executed. Their effect on the performance utility of the task process is then evaluated. The reward associated with the outcome will increase if the utility of the task process increases following the change. Otherwise, the reward decreases.

One of the possible solutions to increasing performance visibility and explicitly linking it to the task processes in such systems is to enforce their integration by applying metamodeling.

5 RASF METAMODEL

The RASF metamodel proposed in this paper is aimed at encompassing models of different kinds of requirements: functional (task process) and nonfunctional (performance), which form the foundation of the software system information structure. The task processes are scheduled by the RACG supervisor and are decomposed into individual tasks. The performance of the task processes has to conform to RAS policies, specifically those on synchronization and communication. This is controlled by first collecting measurement data on those processes, and then analyzing the data according to the decision criteria, determining the utility of the task process, and taking action intended to further increase task process utility.

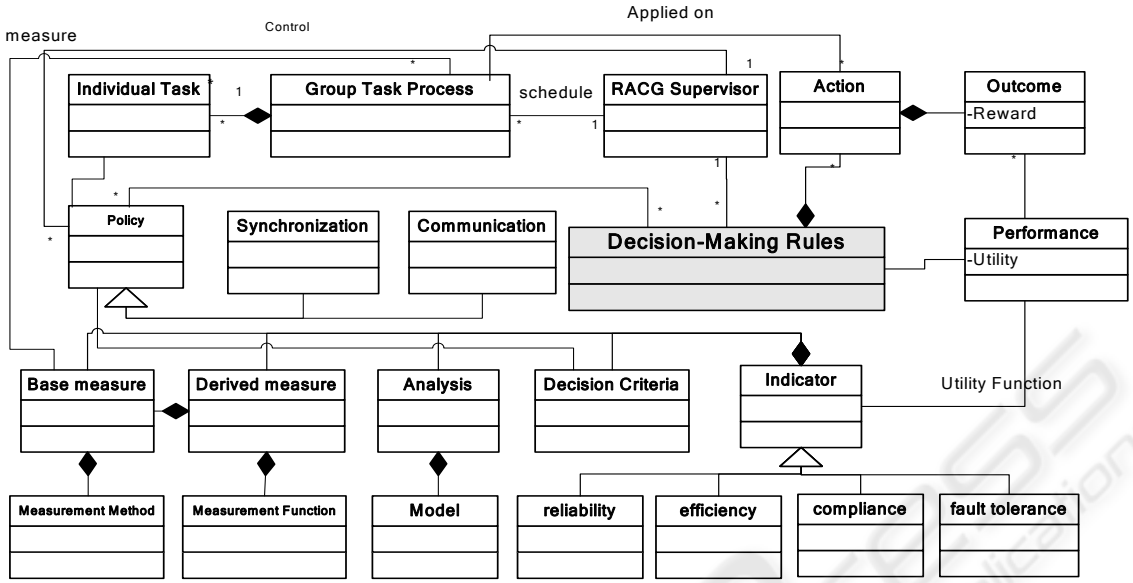

Figure 2: RASF (partial) metamodel.

Knowledge on RAS task processes, the performance hierarchy of measurable characteristics in line with the RAS self-management decision-making policies, and the rules and the relationships among them are represented in the metamodel structure in Figure 2. Figure 2 depicts the semantics of the relationships between different constructs of the RASF Metamodel in terms of a UML diagram. It should be noted, however, that such a diagram does not formally capture the semantics of the constructs or their relationships. This lack of formalism is an obstacle in the process of formalization and consequent automation of the performance modeling and self-management mechanisms. Furthermore, the theoretical validity of the measurement procedures and the decision-making process cannot be established from this semi-formal notation.

This lack of formalism prompted us to exploit the idea of a uniform graphical formalization of the metamodeling of RAS behavior and performance self-management within the RAS life cycle based on the representational theory of measurement and category theory.

6 RASF METAMODEL WITH CATEGORY THEORY

Category theory is a branch of mathematics that, par excellence, addresses "structure", which is the main motivation behind this research: the structure emerges from interactions between elements as captured by arrows, and not extensionally as in set theory. Compared with other methods of formalizing software concepts, category theory is not a semantic domain in which the description of components and connectors is formalized, but rather involves the semantics of interconnections, configurations, instantiation, and composition, which are important aspects of engineering RAS with both autonomous and autonomic behavior. Moreover, automation may be achieved using category theory.

Category theory provides the basic building blocks of metamodeling using the fundamental notions of *category*, *object*, *morphism*, and *functor*. Informally, a category may be regarded as a collection of heterogeneous objects and morphisms which model the social life of these objects, that is, their interactions. A category can be defined as zero or more objects bound together, where each object may be either a primitive or a category. In addition, a category may be augmented, diminished, or joined with other categories to produce a new category. Formally, a category consists of objects: *A, B, C*, etc., and arrows (morphisms) $f: A \rightarrow B$ where, for each arrow f, there are given objects: *dom(f), cod(f)* called the domain and the co domain of *f*, and indicated as $A = dom(f)$ and $B = cod(f)$ respectively. Central to category theory is the notion of composition: given arrows $f: A \rightarrow B$ and$: B \rightarrow C$ with $cod(f) = dom(g)$, there is an arrow: $g \circ f : A \rightarrow C$ called a *composite* of *f* and *g*. For each object *A*, there is a given arrow: $id_A: A \rightarrow A$ called the *identity arrow* of *A*. The category must satisfy the following

laws: i) Associativity: $h \circ (g \circ f) = (h \circ g) \circ f$ for all $f: A \rightarrow B$, $g: B \rightarrow C$, $h: C \rightarrow D$; ii) Unit: $f \circ id_A = f = id_B \circ f$ for all $f: A \rightarrow B$. A functor $F: C \rightarrow D$ between categories C and D is a structure-preserving mapping of objects to objects, along with arrows to arrows: i) $F(f: A \rightarrow B) = F(f): F(A) \rightarrow F(B)$; ii) $F(g \circ f) = F(g) \circ F(f)$; iii) $F(id_A) = id_{F(A)}$

Category theory for RAS self-management has adopted an approach of correction by construction, through which components are specified, proved, and composed so as to preserve their properties. In an abstract sense, we are dealing with arrow diagrams of task processes where the existing arrows represent cooperation channels in a very general way. This gives us the justification for associating the Task Process category with the PATH category, as described in (Pfalzgraf, 2004), where the morphisms are sequences (paths) of consecutive arrows, each node representing a task and each arrow being a structure-preserving mapping, that is, a morphism. This defines the composition of arrows in a natural way (concatenation of consecutive arrows), and this composition is associative. The identity arrow with respect to each object in *Task Process* will be assumed to exist by definition; according to graph theory, it is a loop to the corresponding node.

The *Task Process* category represents the empirical relational structure in the measurement procedure, and it includes the task categories and their relations. The *Base Measure* category represents the numerical relational structure to which the empirical relational structure is mapped. The measurement procedure is deemed valid if it is a structure-preserving (homomorphic) function. It is easy to see that the mapping *Measurement Method (MM): Task Process → Base Measure* between the categories *Task Process* and *Base Measure* is a functor of objects to objects along with arrows to arrows satisfying the functor property outlined earlier. The base measures can be further combined using categorical products and mapped to derived measures (characterizing different performance characteristics) by a morphism *Measurement Function (MF): Base Measure* x *Base Measure* x *...→ Derived Measure*.

In software engineering decision making, we often consider multidimensional alternatives with a variety of quality characteristics or from several points of a view. Such situations arise when we are trying to explain a dependent variable, such as performance utility, on the basis of a number of independent variables, such as reliability, fault tolerance, etc. In order to calculate a utility function of multidimensional alternatives, we need to define a collection of alternatives. We think of the set of alternatives as a Cartesian product of all considered attributes characterizing performance expressed as *Reliability* x *Fault Tolerance* x *Efficiency x Compliance*, and the set of decision criteria, where *Reliability* is a set of all possible values of the domain of the utility function morphism for reliability, and so on. The categorical product relations p0, p1, p2, p3, and p4 are the corresponding projections of the product *Reliability* x *Fault Tolerance* x *Efficiency* x *Compliance* x *Decision Making Criteria* to *Decision Making*, *Reliability*, *Fault Tolerance*, *Efficiency and Compliance* respectively (see Figure 3). Such a product corresponds to all possible alternatives representing the values of the *Indicator*.

The functor *Utility Function: Reliability* x *Fault Tolerance* x *Efficiency* x *Compliance* x *Decision Making Criteria → Performance Utility* in Figure 3 maps the Indicator alternatives to a sample scale where performance utility is qualitatively categorized as Excellent, Acceptable, or Unacceptable. These performance categories provide feedback on performance utility, and help determine whether or not the task processes in the RAS satisfy the performance policies or need improvement. The functors *Method (MM)*, *Measurement Function (MF)*, and *Utility Function* have to satisfy the postulates of the representational theory of measurement. By definition, each functor is a structure-preserving mapping and thus guarantees the theoretical validity of the performance assessment.

We model performance self-management as a decision-making process in category theory as sequences of consecutive arrows linking *Performance Utility* to *Actions* to *Outcomes*. The generic functor *Decision Rule: Preference Utility → Actions* maps each object of *Preference Utility* to an object in *Actions*. Each action has to be mapped to an outcome, or set of outcomes, and each outcome is associated with a reward that affects the decision criteria (or policies). The outcome is meant to improve task process performance, and the execution of the prescribed changes is modeled with the generic functor *Execute: Outcomes → Task Process*.

The diagram in Figure 3 can be abstracted as a concatenation of consecutive arrows *Measurement Method (MM)*, *Measurement Function (MF)*, *Utility Function*, *Decision Rule*, *Execute*. The diagram commutes, which guarantees reliable self-assessment on task process performance and valid decision making based on performance utility.

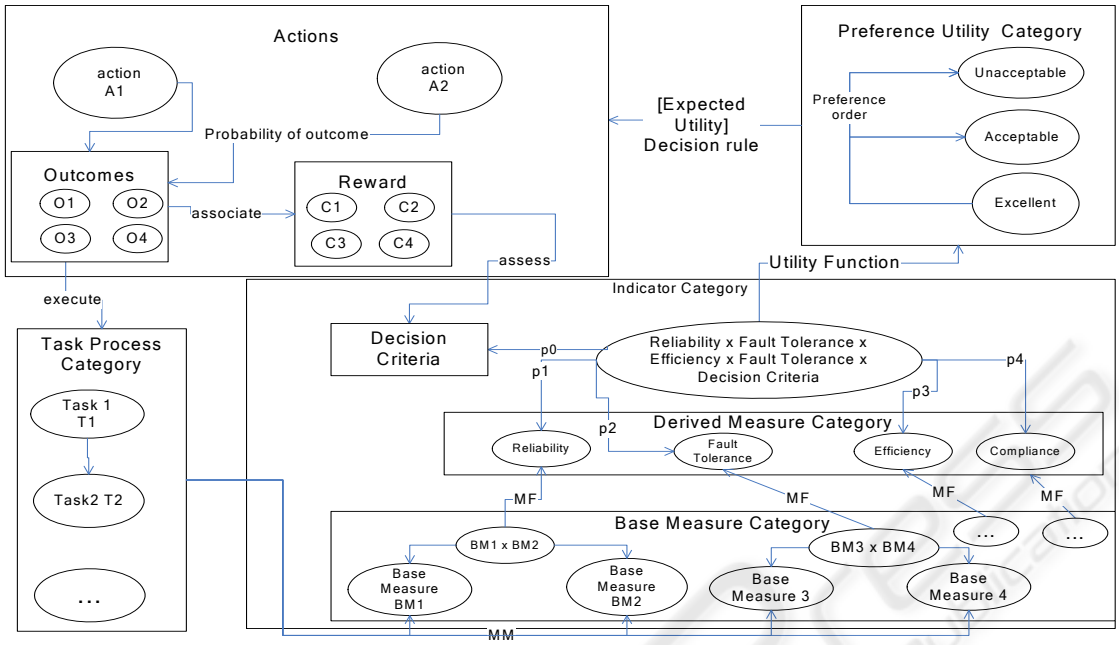

Figure 3: Categorical representation of RAS metamodel with built-in performance self-monitoring.

7 CONCLUSIONS AND FUTURE WORK

The research reported in this paper proves that category theory can provide a sound, scientific, and theoretically valid basis on which to integrate the RAS structure and performance models, which can be treated as mathematical objects and investigated by formal analysis. It should be noted that the concepts of soundness and completeness do not arise in categorical specification. We are currently working on the development of a graphical tool to capture RAS modeling and performance measurement through category theory.

Given the behavior of the rapidly evolving RAS and the need to effectively make decisions during runtime, there is a need to propose probabilistic analogs for traditional algebraic theories of fundamental measurement. We aim to achieve this goal by providing a probabilistic theory of software measurement which can act as a foundation for both measurement and decision making in complex RAS processes. One of the extensions to the current RASF formalization we are now investigating is the use of the Markov Decision Process for establishing the level of confidence in the choice of decision-making rules in a given context. The choice of actions in the decision-making process can be made on the basis of expected utilities and on the analysis of the probabilities associated with each alternative outcome. This issue will be tackled in our future work.

REFERENCES

IBM Corporation, 2006. An architectural blueprint for autonomic computing. *White Paper*, 4th Edition.

IBM Tivoli, 2005. Policy Management for Autonomic Computing – Version 1.2. *Tutorial*, IBM Corp.

ISO/IEC 9126-1:2001 International Standard, 2001. Software engineering – Product quality – Part 1: Quality model.

Abdelwahed, S., Kandasamy, N., 2006. A Control-Based Approach to Autonomic Performance Management in Computing Systems. In *Autonomic Computing: Concepts, Infrastructure, and Applications*, pp. 149-167, CRC Press.

International Standard ISO/IEC 15939 Second Edition, 2007. Systems and software engineering — Measurement process.

Roberts, F., 1979. Measurement Theory. *Encyclopedia of Mathematics and its Applications*, Addison-Wesley.

Pfalzgraf, J., 2004. ACCAT tutorial. Presented at *27th German Conference on Artificial Intelligence (KI-2004)*, September 24, 2004: http://www.cosy.sbg.ac.at/~jpfalz/ACCAT-TutorialSKRIPT.pdf