Creating complex, adaptable management strategies via the opportunistic integration of decentralised management resources

Abstract

The ambitious goals of autonomic management require complex, adaptable processing capabilities that prove extremely difficult to conceive and implement. This paper proposes a solution for the opportunistic integration of specialised autonomic management resources, so as to obtain complex, adaptable management strategies. The paper introduces an architecture that follows the proposed solution and provides a reusable framework that implements this architecture. The solution's validity is indicated by experimental results obtained by testing the framework prototype on a sample home security application.

1. Introduction

While autonomic management solutions become critical for business success [6], the capabilities they must exhibit make them difficult to conceive and implement (e.g. [5], [3]). Available solutions generally focus whether on addressing specific management concerns (e.g. runtime monitoring, pattern recognition, or optimal configurations), or for administering a certain application, running on a given platform, with respect to a certain business goal. Nonetheless, little reusable support is currently available for integrating existing solutions and facilitating the development of complete autonomic applications. We consider that the progress of Autonomic Computing critically depends on the ability to capitalise on existing results. Autonomic management functions are complex due to their ambitious goals. Implementing autonomic management behaviours involve multiple complicated tasks, at various abstraction levels. These include collecting heterogeneous data, interpreting information, diagnosing problems and providing solutions while avoiding conflicting management plans. Additionally, autonomic managers must adapt their administrative strategies to react to changes in managed resources, accumulated knowledge, running environments and business objectives.

Using traditional software development approach for addressing this problem leads to difficulty. This approach is based on a centralised, top-down view of system conception and design, where developers fully specify and control the overall application behaviour (e.g. [11] or [4]). This view raises difficulties when behaviours are laborious to understand and define in a single specification. When autonomic managers are required to react to unpredictable scenarios, the space of detectable conditions and desirable decisions grows exponentially. Predict all possible situations and provide all necessary solutions in a manager's logic is difficult.

A much less exploited approach, advocated in this paper, is to build complex strategies from simpler, specialised components that dynamically collaborate to solve complicated, possibly unexpected problems. In this view, specialised *management resources* (MRs) are dynamically identified and assembled for detecting and solving diverse administrative challenges. Available MRs *collaborates opportunistically* depending on the current situation, requirements and available information during runtime. A precise, exhaustive specification of monitoring, analysis, planning and execution directives is no longer required. An architecture following this approach has been presented in our previous work [3]. It allows different management concerns to be implemented in isolation by providing mechanisms to develop, compose opportunistically and manage the lifecyle of reusable MRs.

The current publication refines the associated concepts and definitions and brings several notable extensions that became necessary at the architectural level. A conflict resolution mechanism for dealing with alternative or diverging management resources is introduced. The presented experimental results show how the extended framework prototype was able to handle more complicated management scenarios on a realistic sample application. In addition, important scaling concerns and a hierarchical organisation of MRs for addressing related challenges are discussed.

2. Proposed solution and general architecture

The architecture aims at obtaining complex, adapted management strategies via the collaboration management resources (MRs). Complex management strategies are developed from simpler tasks (monitoring a parameter, detecting a problem type, planning a specific solution), separately performed by individual MRs. MRs'integration is opportunistic: determined during runtime, depending on the context (conditions and requirements). Integration involves the separate concerns of collaboration, communication and control.

The architecture was divided into two layers [Figure 1]. First, the *Resource Management layer* consists of the available management resources and of the shared communication channel. Second, a *Management Adaptation layer*

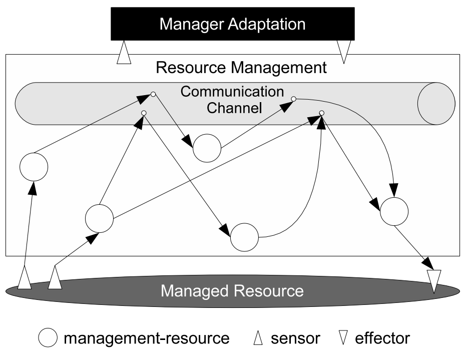

Figure 1. Proposed architecture overview

supervises and reconfigures the Resource Management layer. At present, the Management Adaptation layer observes active MRs in the Resource Management layer and changes their combination. For example, a MR identified as faulty or inefficient is dynamically replaced by an alternative management resource by changing the corresponding subscriptions to relevant topics. Additionally, the Management Adaptation layer can be extended to dynamically search and retrieve MRs from external repositories.

MR are specialised for processing certain data types. Input data from the managed environment arrives into autonomic managers via specialised monitoring MRs. No negotiation or explicit collaboration protocol exists amongst MRs. The focus of control of the application is determined by the activation of MRs in response to the available information. Data is transmitted via a shared *communication channel* [Figure 1]. Within this channel, data is organised into *topic of interest* (or topic) corresponding to a management concern (e.g. problem type occurrence, a certain parameter value). The communication channel uses a publish/subscribe model, which delimits the area of interest of each management resource. MRs subscribe to topics they can interpret and publish their results to topics affected by the data types they produce. They collaborate by reacting to each other's results.

Alternative or redundant MR addressing the same management concern might be activated at the same time. For example, multiple planning resources may be introduced to deal with the same problem type. The *conflict resolution* function assigned to the shared communication channel is hence critical. The current conflict resolution method is based on an inhibition mechanism : MR activation inhibits the activation of conflicting MRs. Inhibition are characterised by a priority (attached to communicated data) and a validity period. It is planned to support more complicated inhibition/activation mechanisms.

Manager are administrable : configurations can be set by system administrators and/or other managers, whether initially, or during runtime. Principal settings are the inhibition parameters and the publish/subscribe configurations that associate management resources to relevant topics.

The strengths of the architecture stem from its high modularity, loose-coupling, separation of concerns and opportunistic control. Implementing specialized tasks as MRs facilitates development and encourages reuse. MRs communicates via a loosely-coupled publish/subscribe model enabling seamless evolution of management logic by refining the set of active MR at runtime. MRs collaboration is separated from their implementation reducing the cost of autonomic management behaviour modification. These features endow the architecture with great flexibility, which renders it highly adaptable, extensible and robust. Iterations are possible before finding a configuration that leads to the desired behaviour. Such procedures would be difficult and costly to perform in a more rigid architectural setting.

3. framework architecture

3.1. Framework Overview

A concrete framework prototype that follows the proposed architecture was developed and validated via several management scenarios on an actual application. It shows a manner of assigning administrative tasks to MRs, realising the communication channel and organising exchanged data types into topics of interest. As previously indicated, the framework architecture is divided into two main layers. More details on the roles and functions of the two layers are available from [3]. The following subsections detail some of the most important aspects of the framework design.

3.2. Topic-based Collaboration

In the Resource Management layer, management concerns are grouped into classes (e.g. monitoring, analysis, planning and execution). Data transmitted via the communication channel is routed via corresponding topics, further organised into specific sub-topics. Exchanged data is characterized by generic meta-information (e.g. unique identifier, producer's identity or processing time [3]) and topic/producer specialized information(e.g. CPU value produced by CPU monitor). The communication channel maps incoming data to affected topics and maintains a list of management resources that are ”interested” in receiving data from that topic.

Building a MAPE ([6]) manager could be done as follows. MRs specialised in resource supervision monitor the managed system and its execution environment, enabling the manager to stay up-to-date. Monitoring MRs publish collected data via corresponding monitoring topics (or subtopics). Subsequently, MRs specialised in data analysis are automatically activated by the arrival of monitoring update-sAnalysis resources detect current problems and publish them to relevant topics. Planning resources are automatically activated by the occurrence of new problem specifications.

Plans are made available via specific execution topics. Finally, execution MRs are activated for performing the necessary actions on the affected managed resources.

3.3. Communication Channel

The communication channel ensures the dynamic integration of MRs and the conflict resolution. It currently uses three mapping tables for distributing data to MRs. First, a *Topic Mapping Table* defines classical publish/subscribe topic subscription.Second, a *Conflict Mapping Table* identifies conflicting MRs for each MR. An expiration duration states the time MR should be inhibited when its conflicting counterpart is activated. Finally, an *Inhibition Status Table* shows the inhibition status of each MR. When a MR is inhibited, priority and expiration time is reported in the table. More than one inhibition may concurrently exist for a single MR entry when different MRs place inhibitors with overlapping periods. (the highest priority is considered at each) Mapping tables are initially set by system administrators and refined by Adaptation Layer at runtime.

The channel uses the Topic Mapping Table to determine the MRs affected by topics. For each of these MRs, the channel uses the Inhibition Status Table to verify whether subscribing MRs are blocked. If the MR is blocked, the priority of the incoming data is compared with the priority of the blocked resource inhibitions. If the incoming data has a higher priority, the inhibition is ignored. Otherwise, the affected MR is discounted. The conflicting resources are blocked, in accordance with the Conflict Mapping Table.

Incoming Data is synchronized with respect to a predefined clock : the communication channel collects all incoming data during a predefined period. Collected data relevant to a topic is aggregated into a separate data set, which is sent to the affected MRs at the end of each period. It ensures that MRs receive enough data to perform their tasks. Various possibilities deserve further study and tradeoffs evaluation.

3.4. Scaling Extentions

Implementing the communication channel in a centralised fashion would rapidly cause scaling difficulties. As the numbers of MRs and topics grow, the communication channel would encounter increasing difficulties in finding the correct entries in the corresponding mapping tables. In addition, this process would be inefficient, since certain MR would be used more frequently than others and as certain resource pairs would seldom, if ever collaborate. Finally, the Adaptation layer would equally present scaling problems when observing and modifying large numbers of MRs and topics.

The architecture was extended to support a hierarchical organisation. MRs are grouped into specialised categories, which are hierarchically organised and identified. Categories and subcategories are related to the data types that MRs can process. For example, different categories can be specified for monitoring, analysis, planning and execution MRs. If the number of monitoring MRs increases, further subcategories can correspond to: all monitoring MRs that extract information on hardware (e.g. memory, CPU and disk),or on software components (e.g. number of instances).

MRs can be *Simple* or *Composite*. Simple MRs implement actual administrative tasks. Composite MR contain their own communication channel and their own set of MRs (Simple and/or Composite). MRs collaborate in the same manner described before. The communication load is distributed among multiple communication channels. As before, MRs return their results to the communication channel they are attached to. The implemented procedure is similar to that of packet routing in IP networks. First, the channel uses the data type prefix to determine whether or not the incoming data affects its own category of MRs. If it does, the channel uses its mapping tables to activate the MRs. Otherwise, data is forwarded to a default channel (e.g. equal or superior).

3.5. Dynamic Adaptation

The Management Adaptation process aims at optimising the administrative behaviour of the Resource Management layer. It uses the current high-level goals and the behaviour of active MRs to dynamically reconfigure the Resource layer. It intervene for recovering from an erroneous situation : it may observe that an active MR fails to reach a conclusion and decide to replace it.

The Management Adaptation layer is designed to follow the hierarchical organisation of the Resource Management layer (Composite MRs). The abstraction level of each Management Adaptation instance corresponds to its position in the hierarchy. At the very top, an instance coordinates the actions of lower-level instances and ensures communication with the external environment. More details on the Management Adaptation layer are available from [3].

3.6. framework implementation

A prototype of the presented framework was implemented following a Service Oriented Component (SOC) approach [10]. This approach capitalises on the advantages of both component and service-oriented paradigms. The loose-coupling and the dynamic discovery characteristics of SOC are indispensable to our architecture. The large number of interconnected elements specific to the framework requires the use of an efficient communication infrastructure. This consideration renders Web services as an unlikely candidate, due to their inherent communication and processing costs. The iPOJO[1] /OSGi[2] Service Oriented

1. iPOJO Project: www.ipojo.org
2. OSGi Consortium: www.osgi.org

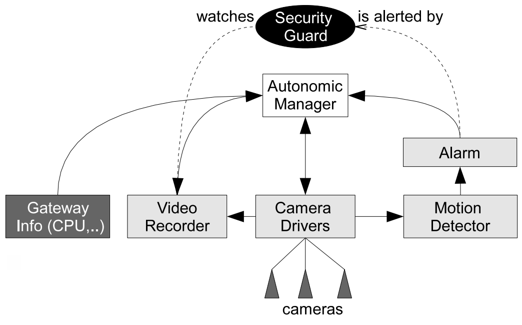

Figure 2. Experimental home security application

Component technology (Java implementation) was adopted for developing the current prototype. iPOJO provides important OSGi extensions, including distributed communication, automatic and dynamic service discovery/binding, and autonomic facilities [2]. Hence, each MR is implemented as an iPOJO service component. The communication channel represents another, composite iPOJO service. MRs are connected to the channel via dynamic iPOJO bindings. The initial composition of a framework instance is specified via a special-purpose XML file (e.g. management resource types, instances and initial mappings). Further framework design and implementation details are available from [3].

4. Sample application and experimental results

4.1. Home Security Application

An experimental application was used for validating the framework prototype. The application was implemented based on our team's experience with the ANSO[3] Project. It supervises a home and notify a security company when an intrusion is detected. The implementation follows the design depicted in figure 2. Several video cameras are employed for capturing images from different rooms. They communicate with the application via specific drivers. Images are sent to a motion-detector, which searches for differences that may indicate movement. In parallel, images are sent to a persistence service, which saves them to a local file system. Guards notified by alarm may first analyse the available images before intervening.

The application runs on a home gateway executing multiple applications. The goal of the autonomic framework is to ensure the functioning of the security application despite fluctuating gateway resources. Hence,scenarios focus on the video cameras and image storage service management, depending on the current application state (i.e. normal or alarm) and on the available disk and CPU resources.

3. ANSO ITEA Project: anso.vtt.fi

4.2. Framework Instantiation and Configuration

The framework instantiation consisted of four Composite MRs - monitoring, analysis, planning and execution, linked in a "classic" control-loop. The monitoring Composite uses three MRs observing the alarm service state, the CPU utilisation and the disk consumption levels. These MRs produce data of types: *monitor.alarm*, *monitor.CPU* and *monitor.disk*.

The analysis Composite uses four MRs for processing monitored data. The first MR can detect presence or absence of alarm. The others detect the crossing of different predefined thresholds by the consumptions of disk and CPU. These thresholds MRs are instances of the same implementation but configured with a different threshold and activated by different topics. One is activated by *monitor.CPU* with threshold set to 95% of the CPU capacity. The other two analysers are activated by *monitor.disk* data with thresholds set at 80% and 90% of the disk capacity. The four analysis MRs produce results of different data types and different priorities: *analysis.alarm* (priority 2) and *analysis.non-alarm* (priority 4) - indicating presence/absence of alarm; *analysis.CPU* (priority 3), *analysis.disk1* (priority 1) and *analysis.disk2* (priority 1) - the crossing of thresholds.

The planning Composite contains four MRs. The Cam_Planner MR decides to switch on/off cameras located in rooms with no access points, depending on the alarm state. It is activated by the *analysis.alarm* or *analysis.non-alarm* topics and issues *plan.camera-on* and *plan.camera-off*. The Standard_DiskPlanner MR, activated by *analysis.disk1* topic, decides to delete 40% of the stored image.

Alternatively, The Alarm_DiskPlanner resource can be activated by *analysis.alarm*, *analysis.disk1*, or *analysis.disk2* topic. Its decision depends on two possible data type combinations. When *analysis.alarm* and *analysis.disk1* data types occur simultaneously, the MR order registered images compression. However, when disk usage approaches the maximum, the planner tries to save the most recent images. Therefore, when activated by the concurrent occurrence of *analysis.alarm* and *analysis.disk2* data, the same Alarm_DiskPlanner decides to delete 40% of the oldest image records. The Alarm_DiskPlanner only proposes a disk management action in case an alarm is present; this planner remain inactive in the absence of an alarm . The default inhibition state of the Alarm_DiskPlanner was set to 2. This enables data of type *analysis.alarm* (of priority 2) to activate this MR. However, it prevents *analysis.disk1* and *analysis.disk2* (of priorities of 1) to activate this MR alone. When an alarm activates Alarm_DiskPlanner, the MR also receives disk related data. Future works will extends the communication to support logical expressions simplifying these settings.

The last alternative is AlarmCPU_DiskPlanner activated by *analysis.alarm*, *analysis.disk1*, *analysis.disk2* or *analysis.CPU*. It remains inactive as long as there is no CPU overload, but once activated receive all data affecting its decisions (i.e. CPU, disk and alarm). For this reason, this planner's default inhibition state was set to 3(priority of *analysis.CPU topic*). When activated, this planner order the deletion of one in three recorded images. This planner preserve CPU consumption avoiding image compression When second disk threshold is crossed, this MR orders the deletion of 40% of images.

Some planning MR decision are conflicting. These conflict are managed by the aforementioned conflict tables. The Cam_Planner is not in conflict with any of the other planners and never gets blocked. The three others planning MR accordingly try to mutually inhibit each other, whenever active, using the maximum priorities of the data that activated them. Based on the priorities of analysis data, the AlarmCPU_DiskPlanner can inhibit the Alarm_DiskPlanner, which can inhibit the Standard_DiskPlanner. Four execution MRs were necessary for the execution Composite for managing cameras activation, deleting a certain quantity of old images,erasing one image out of every given number, and compressing recorded images.

4.3. Management Scenarios and Obtained Results

The fully implemented application works with real video cameras. However, for better controlling the scenarios, experimental drivers simulate the existence of multiple cameras. The results are displayed in Figure 3. The graph shows the percentages of CPU and disk consumption, the starting and stopping of alarms and the alternate activation of the four execution MRs.

The starting and stopping of alarms was sensed by the alarm monitoring MR and triggered the alarm analysis MR. This MR produced *analysis.alarm* (or *analysis.non-alarm*) data which triggered the Cam_Planner and resulted in data of types *plan.alarm-on* (or *plan.alarm-off*). The graph shows the activation of the camera execution resource almost superposed with the starting and stopping of application alarms (e.g. at times 120s and 380s for starting; and 210s and 460s for stopping). When more cameras are activated (at 120s and 380s), the slope of the disk consumption curve increases : more images are stored. When the alarm was off, disk overloads were handled by the Standard_DiskPlanner ordering deletion (times 215s, 275s and 330s).

On alarm(*analysis.alarm*) and first disk threshold (*analysis.disk1*),the Alarm_DiskPlanner was activated. The planner ordered compression activating the compression execution MR. The activation of this execution resource occurred 5 times during the first alarm period, between 120s and 210s. This strategy became less and less efficient as the sizes of already compressed images could no longer be reduced. A scenario in which in the Adaptation layer intervened to tune the choice of the planning MR was presented in [3]. As the

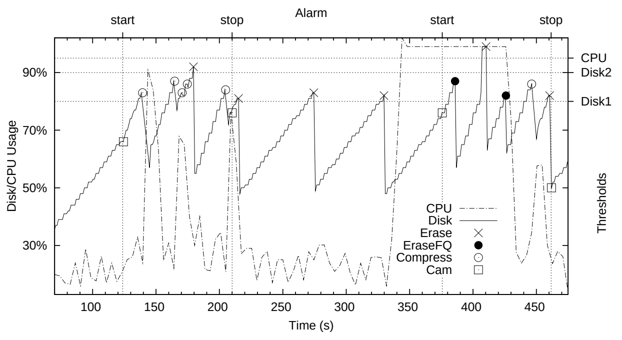

Figure 3. Disk management in a security application

compression increased CPU consumption, it was avoided during CPU overloads (e.g. between 330s and 430s).

On alarm and second disk threshold (*analysis.disk2*) , the Alarm_DiskPlanner order deletion instead of compression. This image deletion by execution MR is shown at the approximate time of 180s. The same MR was activated at time 410s, but this time ordered by AlarmCPU_DiskPlanner. This occurred when data of type analysis.alarm, analysis.disk2 and analysis.cpu was simultaneously present in the communication channel. However, when the disk consumption was in-between the two thresholds (i.e. presence of analysis.alarm, analysis.disk1 and analysis.cpu data), the AlarmCPU_DiskPlanner chose the selective image deletion strategy instead, activating the corresponding execution MR (*plan.step-delete*). This scenario occurs twice on the graph, during the second alarm period, when the CPU threshold is also crossed (i.e. at times 390s and 425s). The CPU overload was induced by a processor-intensive application.

These results show how the proposed framework can handle various administrative scenarios by dynamically creating management strategies from individual management resources of various types. The flexibility of the adopted solution allowed the seamless definition of behaviours for handling diverse combinations of external conditions.

5. Related works

The importance of applying the service-oriented paradigm to autonomic management applications is reflected by the publication of specific Web services standards[7][8] shows the advantages of standard interfaces for autonomic elements, as it allows the creation of autonomic applications from services developed by multiple providers. The availability of such interfaces is vital for building adaptable autonomic managers with dynamically interchangeable elements.

In the autonomic computing field, several projects have started to develop generic architectures, engineering principles and execution platforms with reusable capabilities that facilitate the creation of autonomic applications (e.g. IBM

Autonomic Computing Toolkit [4], Autonomia [5], AutoMate [6], BioNets[7], Amorphous Computing[8], Autonomic Networked Systems[9] or Recovery-Oriented Computing [10]).

Other research areas are concerned with the development of automatic reasoning functions. Such areas include Artificial Intelligence, Robotics and Automated systems. Most significantly, concepts related to Multi-Agent Systems (e.g. [12]) and Blackboard architectures (e.g. [9] or [1]) seem most tightly related to our approach. Nonetheless, in Multi-Agent Systems, agents are autonomous entities capable of identifying and of negotiating with peer agents in order to form necessary collaborations. In our approach, collaborations emerge from the simple reactions of management resources to the occurrence of data they can interpret. This behaviour closely resembles that of the Blackboard model and consequently features similar advantages and difficulties. In contrast to agents, the autonomous capabilities of MRs are quite reduced, with the channel providing basic communication facilities.

6. Conclusion and future perspectives

This paper proposed a solution for building complex, adaptable autonomic management applications via the opportunistic collaboration of decentralised management resources. The associated architecture enables management resources to form flexible, reactive collaborations by communicating data through a shared, topic-based channel. Additionally, the architecture introduces a second autonomic management layer, for dynamically adapting the collective behaviour of management resources, based on previously obtained results and on the priorities of current business goals. The proposed architecture addresses scalability issues by organising management resources into hierarchical categories. The framework's autonomic management capabilities were demonstrated in several administrative scenarios on a realistic home application. Obtained results showed the potential of the proposed approach to support the development of complex, adaptable management behaviours via the collaboration of specialised MRs.

Immediate research efforts will focus on iteratively assessing, refining and extending the framework's capabilities on increasingly more complex management scenarios and applications. Some of the already identified extensions include support for logical operations in the communication channel and for hierarchical organisations with Simple and Composite management resources. In addition, future perspectives will target the development of the Management Adaptation layer, with dynamic discovery and integration of management resources (i.e. services) when available MR prove insufficient for handling current challenges.

4. Autonomic Computing Toolkit (IBM developerworks): www.ibm.com/developerworks/autonomic/overview.html

5. Autonomia (University of Arizona): www.ece.arizona.edu/~hpdc/projects/AUTONOMIA

6. AutoMate (Rutgers University): automate.rutgers.edu

7. The Bio-Networking Architecture (University of California Irvine): netresearch.ics.uci.edu/bionet

8. Amorphous Computing: swiss.csail.mit.edu/projects/amorphous

9. Autonomic Networked Systems (ANS) (Imperial College): www.doc.ic.ac.uk/~asher/ubi/ansproj

10. Recovery Oriented Computing (ROC) project (Berkley and Stanford Universities): roc.cs.berkeley.edu

References

[1] D. D. Corkill. Blackboard systems. *Artificial Intelligence Expert*, September 1991.

[2] A. Diaconescu, J. Bourcier, and C. Escoffier. Autonomic ipojo: Towards self-managing middleware for ubiquitous systems. *International Workshop on Social Aspects of Ubiquitous Computing Environments (SAUCE)*, 2008.

[3] A. Diaconescu, Y. Maurel, and P. Lalanda. Autonomic management via dynamic combinations of reusable strategies. In *Second International Conference on Autonomic Computing and Communication Systems, Autonomics 2008*, Turin, Italy, September 23 - Sep 25 2008. ICST.

[4] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[5] M. C. Huebscher and J. A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28, 2008.

[6] D. M. Kephart, Jeffrey O. et Chess. The vision of autonomic computing. *Computer*, 36, 2003.

[7] H. Kreger and T. Studwell. Autonomic computing and web services distributed management, 2005. www.ibm.com/developerworks/autonomic/library/ac-architect/.

[8] B. Miller. The Standard way of autonomic computing, 2005. www-128.ibm.com/developerworks/autonomic/library/ac-edge2/.

[9] H. P. Nii. Blackboard systems, part one: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine 7(2)*, pages 38–53, 1986.

[10] M. P. Papazoglou and D. Georgakopoulos. Service Oriented Computing. *Communications of the ACM*, 46:25–28, October 2003.

[11] S. Sicard, F. Boyer, and N. D. Palma. Using components for architecture-based management: the self-repair case. *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 101–110, 2008.

[12] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. A multi-agent systems approach to autonomic computing. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 464–471, Washington, DC, USA, 2004. IEEE Computer Society.