Rule Engine Based Lightweight Framework for Adaptive and Autonomic Computing

Jakub Adamczyk, Rafał Chojnacki, Marcin Jarząb, and Krzysztof Zieliński

Institute of Computer Science, AGH - University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
{j.adamczyk, chojnacki, mj, kz}@agh.edu.pl
<http://www.ics.agh.edu.pl>

Abstract. The paper describes a framework architecture called the Autonomic Management Toolkit (AMT). This toolkit was implemented to support dynamic deployment and management of adaptation loops. This requires automatic resource discovery, instrumentation and attachment to Autonomic Manager (AM), and furthermore a scalable and easily changed decisionmaking module, which is a major part of the AM. The architecture of a system satisfying these requirements is proposed and described. This system is compared to PMAC (Policy Management Autonomic Computing) – a highly advanced software tool offered by IBM. The central element of AMT is a lightweight AM with Rule Engine as a decisionmaking module. This makes the proposed solution lightweight and flexible. The AM activity is very briefly specified and the process of constructing an execution loop is described. The proposed interfaces are specified. These interfaces are generally sufficient to support a wide range of policies, including standard regulators, well known from control theory. Subsequently, AMT usage is illustrated by a simple example. The paper ends with an overview of related work and conclusions.

Keywords: autonomic manager, rule engine, adaptive, workload management, Drools, PMAC.

1 Introduction

Adaptive and autonomic management of computing resources is a well known problem facing computer scientists. For decades system components and software have been evolving to deal with the increased complexity of system control, resource sharing, and operational management. The evolution of these trends addresses the increasingly complex and distributed computing environments of today [2]. The research started several years ago by IBM, Motorola, SUN and many other companies, has resulted in software environments supporting policy-driven system management for autonomic computing [6]. One of those environments is PMAC (Policy Based Autonomic Computing) [5], developed by IBM. The majority of existing [5, 6, 17] policy-driven systems rely on the construction of an adaptation loop.

A challenging problem is how to construct a system able to discover new resources during runtime, automatically generate an intermediate management layer and apply

selected policies with limited human intervention or through a software system re-start. Another related issue is the construction of a policy evaluation engine. The decisionmaking module, being a fundamental part of the Autonomic Manager, should be powerful enough to process many policies in a scalable way and should accept their specification expressed using existing rules [4] or policy specification languages [18]. Implementation of a system satisfying these requirements opens new possibilities for building a management or adaptation system loop dynamically and with limited effort. This was the main motivation behind the construction of the Autonomic Management Toolkit (AMT), described in this paper.

AMT is a lightweight framework for constructing adaptive and autonomic computing systems. The proposed solution exploits a different concept in comparison to the complex and heavy PMAC technology and promotes a lightweight approach [1]. Lightweight development is such an extensive topic that often that it is difficult to specify exactly what it means. In this paper, a lightweight framework implies programming with Plain Old Java Objects (POJOs) using design patterns implemented with JMX [8] technology MBean components used for coupling objects and integrating autonomic managers. The paper focuses on a framework architecture called AMT (Autonomic Management Toolkit) which uses JMX to control managed resources. The central element of this architecture is the lightweight autonomic manager. The most innovative concept applied in AMT construction is the usage of a Rule Engine as a reasoner within the Autonomic Manager. Such an approach introduces a natural mapping of term policies to a production rule definitions [5]. As this mapping could introduce some constraints related to policy expression, it is not applied in most applications. It performs operations on managed resources registered with the manager. The managed resource's interface is fully compliant with PMAC. Newly discovered resources can be registered during runtime. A wrapper object implementing the managed resource interface for any resource represented as an MBean can be dynamically generated. This guarantees full flexibility and adaptability, not heretofore supported by PMAC. The wrapper objects support local and remote operation invocation and event notification, thus the managed resources can be located anywhere.

The ongoing work is related to previous research performed at AGH UST concerning monitoring and management of virtualized environment [3], [12]. The structure of the paper is as follows. First, in Section 2, AMT architecture is shortly described and the motivation behind its construction is discussed. Next, in Section 3 the functionality of the AMT Autonomic Manager is presented. In Section 4 integration techniques for managed resources are described. In Section 5 examples of AMT usage for management of Solaris containers are presented. Section 6 contains a comparison of existing solutions. The paper ends with conclusions.

2 AMT System Architecture Description

The AMT architecture presented in Fig. 1 consists of several key subsystems, which correspond to components existing in PMAC, but are constructed to provide features listed in Table 1. A single entry point to AMT is the Autonomic Manager interface, which exposes operations supported by software modules of the Autonomic Manager described below:

- Policy Management Module (PMM) – policies defined by system administrator are deployed to AMT.
- Policy Evaluators Module (PEM) – policies are obtained from storage, instantiated by a given reasoner and evaluated. The key point of this subsystem is an interface that supports interoperability with different reasoners.
- Resource Access Module (RAM) – defines resources and manages interactions compatible with AMT specification. This part of the system has been implemented to offer full functional compatibility with PMAC.

These three modules offer the core AMT functionality which is accessible by the AMT Console Module (ACM).

Fig. 1. Key components of architecture of Autonomic Management Toolkit

A JIMS Integration Module (JIM) is also available. This module enables integration of AMT with the JIMS [10], [11] (JMX-based Infrastructure Monitoring System) infrastructure responsible for discovering and instrumentation of managed resources represented as MBeans. This part of system can be substituted by any other monitoring system or instrumentation technique.

Each policy available in AMT has the following properties:

- Name – must be unique within policy scope.
- Scope – hierarchical structure of non-strings used for denotation of the given policy applicability domain..
- Type – similar to PMAC, each policy can be either solicited or unsolicited. Each solicited decision is a direct decision requested from a resource or any other external system. Solicited decisions have an input and an output. The input and the output are sets of key-value mappings. An unsolicited decision is a reaction to a system state change, evaluated periodically. An evaluation property defines the timespan between intervals.

- Evaluation in milliseconds – only for unsolicited policies which are evaluated periodically; evaluation in milliseconds defines the timespan between policy evaluations.
- Activated on startup – if set to true, policy evaluation is started immediately following AMT initialization.
- Reasoner – reasoner’s name, valid for policy evaluation.

AMT is designated to support many Rule Engines running in parallel, which may be attached and detached to a running Policy Evaluator without system restart. To resolve incompatibilities between different Rule Engines, AMT introduces the Reasoner Adapter (RA) concept which implements the interface described in Table 1.

Table 1. Specification of the Reasoner Adapter interface

Method Name	Description
attachResource	Attaches new resource
detachResource	Detaches resource
addPolicy	Adds policy for unsolicited and solicited decisions
removePolicy	Stops policy evaluation and removes it from Rule Engine
evaluateSolicitedDecision	Returns solicited decision or null if decision is not available
getInfo	Returns Rule Engine description
shutdown	Performs required operations upon detachment

3 AMT Autonomic Manager Activities

Application of Rule Engine as a major part of the Autonomic Manager influences its startup time procedure and runtime activity. When the Rule Engine is started, it activates a rule base in the production memory for Rule Engine use. The rule base contains all rules and class definitions to be evaluated against facts. The Rule Engine is driven by the emergence or change of facts residing in the Working Memory. Such changes may be generated by external events, as results of Rule Engine activity or by the expiration of a timer which associated with previously received facts. In our case, facts represent manageable resources, which are implemented as MBeans components created on demand, as described in Section 4.

The activity of the Autonomic Manager is performed in the following steps, representing the execution of an adaptation loop, depicted in Fig. 2:

1. Managed Resources (MRs) are instantiated as MBeans.
2. Resource Wrappers of MRs which play the role of facts, are constructed and inserted into the Working Memory. The Reasoner Adapter interface is used in this step.
3. Production Rules representing policies are loaded into the Production Memory. At this point, the Inference Engine is also started.
4. The Pattern Matching algorithm is performed on all rules in the Production Memory and facts present in the Working Memory.

Fig. 2. Autonomic Manager processing steps

5. All rules that are evaluated as true are added to the Agenda to be performed.
6. Action is performed on the representation MR in the Working Memory.
7. The action is forwarded to MR via the Resource Wrapper and enforced with effectors.
8. MR parameter changes accessed by sensors are communicated to the Resource Wrapper, which in turn triggers execution of step 4.

Steps 4 to 8 constitute the main execution loop of the Autonomic Manager. Since rules are declarative Knowledge Representation forms, they are not analogous functions in a procedural language. Instead they are fired in response to changes in the facts available to the rule engine.

4 Managed Resource Representation

For the execution loop implementation, a crucial point is the representation of managed resources. AMT uses the JMX technology for accessing resources. A key element of JMX architecture is the MBean Server, which is a container for MBean components – Java objects which represent manageable resources and which allow operating on attributes (read/write values), executing actions and subscribing to notifications of events related to these resources. Each MBean registered in a server has a unique assigned name and is accessible for clients using various protocols supported by appropriate connectors (RMI, SOAP, HTML, SNMP) installed in an MBean server.

Resource representation in AMT is based on the PMAC framework concept, thus it also enables the use of the PMAC Autonomic Manager. To meet this requirement we have defined an abstract class *Resource*, which implements the *JavaManagedResource* interface, for resource representation. The resource class specification and its dependencies on other classes and interfaces are depicted in Fig. 3.

Fig. 3. Specification of the AMT resource classes

For each manageable resource, an MBean resource wrapper class is generated with resource-specific action methods. The implementation of such a wrapper class depends on MBean properties, methods and notifications. When an MBean attach request process is initiated, AMT checks if a suitable resource wrapper class is available. If there is no such class then a wrapper generation process is started. The wrapper is generated from a parameterized template and transformed into Java class source code using the Apache Velocity library, then compiled with the Java Compiler API and loaded into the JVM.

The proposed mechanisms of the resource attachment process are flexible and enable discovery of a managed resource during runtime, with standard JMX protocols, and generating a suitable wrapper class. Therefore, the adaptation loop may be constructed on the fly, which is a unique feature of AMT. Building upon our former work [10, 11, 12] on management of virtualized resources, it creates a complete framework which satisfies the aforementioned requirements.

5 Examples of AMT Usage

This section presents a case study of controlling workloads within the Solaris 10 environment. The AMT is used for workload management of Solaris Containers [8] based on mechanisms specified by the *Control Theory* [13] and structured as a closed-loop AM workload manager. The whole system is treated as a black box. The controller uses current CPU usage values and adjusts them by changing shares (control signals) to maintain the requested CPU usage. The implementation uses a control loop managed by the AMT, running within a *JIMS Management Agent* on the machine on which the workload is running. The goal of the control loop was to adjust the *project.cpu-shares* resource control to a value which would assure that a given percentage of CPU time

would always be available in conditions of constant load for a specific workload – for instance, a given project should be guaranteed 70% of CPU time when other active workloads (disturbances) are also present.

A sample controller algorithm could use the *Proportional (P)* regulator – Eq. 1, where: (i) U_w – required usage of CPU by workload W_w , (ii) U_w^t – usage observed at time t by workload W_w , (iii) S_w^t – number of shares set at time t for workload W_w , (iiii) K_p – proportional coefficient.

$$S_w^{t+1} = S_w^t + K_p * e(t), \text{ where } e(t) = U_w^t - U_w \quad (1)$$

The above control algorithm for Solaris 10 is implemented with the Drools rule-based policy. The rule definition implementation is enhanced with some heuristics; this rather complex task is explained in more detail in [12].

The experiment was performed under Solaris 10 running on a Sun Blade B100 (1GB RAM, CPU SPARC 650 Mhz) board. In this scenario a *constant* disturbance was activated after the controlled workload reached a steady state (considering the fact that it was the only CPU-bound workload, it reached close to 100% CPU usage). Fig. 4 presents a case where only one CPU-bound workload is started in the selected project, at its beginning. After a few seconds, when CPU usage increases to almost 100%, two other CPU-bound workloads are started in other projects, which results in a drop of CPU usage of the selected project. Subsequently, after several more seconds, controller P is turned on. It changes the share allocation for the controlled project, stabilizing CPU usage at 70%. The experiment was performed with a sample value of $K_p = 3$.

This section presented a simple scenario of how AMT can be used. It is rather a proof of concept which validates our AMT design than a real-life problem. We have described policies which focus on system optimization and try to maintain the system

Fig. 4. AMT used for adaptive management of Solaris 10 *project.cpu-shares* resource

in a stable state. These policies are able to initiate valid reactions when the environment state changes.

6 Related Work

The AMT system offers an alternative solution to the PMAC framework. Limitations and drawbacks of the PMAC in comparison to the Autonomic Management Toolkit (AMT) are presented in Table 2. This comparison points out the AMT toolkit features which make the proposed solution more flexible and easy to use.

A list of projects in the area of autonomic computing, conducted by academic institutions, can be found in [14]. The presented research can be compared to research performed under the project entitled “*Models and Extensible Architecture for Adaptive Middleware Platform*” [15], by National ICT Australia. Similarly to AMT, this project assumes that the ability to develop/implement autonomic services must have the following general characteristics: (i) Standards-based: programming frameworks to construct autonomic services must leverage standard services; (ii) Maintain Separation of Concerns: The adaptive behavior must be expressed in complete separation from the main application’s business logic; (iii) Deployment/Development Scalability: solutions to provide adaptive behavior should scale down to small-scale applications, and scale up to large multi-server deployments. Despite similar assumptions, the project facilitates the development of adaptive behavior for legacy server applications [16] with support for aspect programming.

The Adaptive Server framework developed under this project [17] supports the development of adaptive behavior for serverside components running on application servers. This is in contrast to AMT, which addresses the general issue of the adaptation loop construction, powered by Rule Engines.

Table 2. Comparison of PMAC and AMT frameworks

Feature	PMAC	AMT
Application server required to support operations	Yes	No
Resource description by WSRF needed	Yes	No
Resources attachment only during the system restart	Yes	No
Policy syntax is verbose	Yes	No
Policy specification expressiveness	Limited	Depends on policy evaluator
Number of policy evaluators available	One	Many
Activity logging and errors reporting	Limited	Full and extendable
Declarative style of policy representation only	Yes	No

7 Conclusions

The implementation of the AMT system exploits the potential of Rule Engine-based computing which is a very attractive solution for policy-driven autonomic computing systems. However, integration of the Rule Engine with managed computer resources is not easy and requires proper virtualization of such resources. This aspect is elaborated in related work [10, 11, 12].

As the Rule Engine [4] is a rather sophisticated software module which supports scalable pattern matching algorithms, it may be used for a large number of facts and rules constituting a representation of knowledge. This feature is important for AC systems typically built as a hierarchy of self-management subsystems. Each such subsystem usually provides [6] self-configuration, self-optimization, self-healing and self-protection functionality. In spite of differences between these terms, they are handled similarly – typically driven by rules, which specify a high-level goal of system activity.

A more exhaustive evaluation of the AMT concept calls for an extensive performance study. The existing overheads and bottlenecks should be identified. The presented work is only a proof of concept, showing solutions to integration and interoperability problems affecting various technologies used for AMT implementation. The constructed framework is fully operational and open to further enhancements.

References

1. Tate, B.A., Gehtland, J.: *Better, Faster, Lighter Java*. O'Reilly Media, Sebastopol (2005)
2. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era, <http://www.research.ibm.com/journal/sj/421/ganek.html>
3. Janik, A., Zielinski, K.: Transparent Resource Management with Java RM API. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 1023–1030. Springer, Heidelberg (2006)
4. Morgan, T.: *Business Rules and Information Systems: Aligning IT with Business Goals*. Addison-Wesley Professional, Reading (2002)
5. Policy Management for Autonomic Computing, Tivoli, version 1.2.1, <http://dl.alphaworks.ibm.com/technologies/pmac/PMDevGuide121.pdf>
6. Strassner, J.C.: *Policy-based Network Management – Solutions for the Next Generation*. Morgan Kaufmann, San Francisco (2004)
7. Sullins, B.G., Whipple, M.B.: *JMX in Action*. Manning Publication Co. (2003) ISBN: 1930110561
8. Lageman, M.: *Solaris Containers – What They Are and How to Use Them*, Sun Microsystems (2005), <http://www.sun.com/blueprints/0505/819-2679.pdf>
9. Szydzio, T., Szymacha, R., Zieliński, K.: Policy-based Context-aware Adaptable Software Components for Mobility Computing. In: EDOC 2006: 10th IEEE International Enterprise Distributed Object Computing Conference, Washington, DC, USA, pp. 483–487 (2006)
10. Zieliński, K., Jarzab, M., Wiczorek, D., Bałos, K.: JIMS Extensions for Resource Monitoring and Management of Solaris 10. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 1039–1046. Springer, Heidelberg (2006)

11. Zielinski, K., Jarzab, M., Wieczorek, D., Balos, K.: Open Interface for Autonomic Management of Virtualized Resources in Complex Systems - Construction Methodology, Future Generation Computer Systems, <http://www.sciencedirect.com/science/journal/0167739X>
12. Jarzab, M., Zieliński, K.: Framework for Consolidated Workload Adaptive Management. In: 2nd IFIP CEE-SET 2007, Software Engineering in Progress, NAKOM, pp. 17–30 (2007) ISBN 978-83-89529-44-2
13. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. Wiley-IEEE Press, Chichester (2004)
14. Portal for Autonomic Computing resources, <http://www.autonomiccomputing.org>
15. Enabling Adaptation of J2EE Applications Using Components, WebServices and Aspects, National ICT Australia, <http://www.cse.unsw.edu.au/~yliu/asf-demo/index.html>
16. Liu, Y., Gorton, I.: Implementing Adaptive Performance Management in Server Applications. In: ICSE 2007 workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS) (2007)
17. Gorton, I., Liu, Y., Trivedi, N.: An Extensible, Lightweight Architecture for Adaptive J2EE Applications. In: International Workshop of Software Engineering and Middle-ware (SEM) (November 2006) (accepted)
18. Kephart, J.O., Walsh, W.: An artificial intelligence perspective on autonomic computing policies. In: IEEE 5th Intl. Workshop on Policies for Distributed Systems and Networks, pp. 3–12 (2004)