Model-Driven Autonomic Architecture

Radu Calinescu
Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Abstract

We present a generic architecture for developing fully-fledged autonomic systems out of non-autonomic components, and investigate how the architecture can be implemented using existing technologies. The universal policy engine at the core of the architecture is configured by means of a model of the resources placed under its control, and uses a set of flexible policies for their management.

1. Introduction

Half a decade since the launch of the autonomic computing manifesto, systems that "manage themselves according to an administrator's goals" [4] are far from ubiquitous. Existing IT components exhibit only limited and isolated elements of autonomic functionality, and autonomic systems will need to handle "legacy" resources for some time to come. Our model-driven autonomic architecture addresses this need by providing a framework for the development of autonomic systems out of non-autonomic components. This differs from other autonomic frameworks that target the management of *autonomic-enabled* components, e.g., applications implemented to take explicit advantage of the interfaces of the autonomic framework [3]. The use of a model of the managed system to configure the universal policy engine at the core of the architecture allows the development of autonomic systems comprising a heterogeneous mix of resources. This is a major improvement over existing approaches, which address the management of a specific type of resource [2, 7].

2. Model-driven autonomic architecture

The autonomic architecture in Figure 1 generalises the author's work on policy-based resource allocation [2], and builds on recent developments in policy and manageability interface standardisation [5, 6], policy expression language design [3] and autonomic system development [2, 7]. The components of the architecture are described below.

Resources A heterogeneous mix of legacy resources and autonomic-enabled resources is managed by the autonomic system. The legacy resources may include physical servers, networks, application servers, virtualisation environments and devices ranging from load balancers and power supplies to PDAs and factory automation equipment.

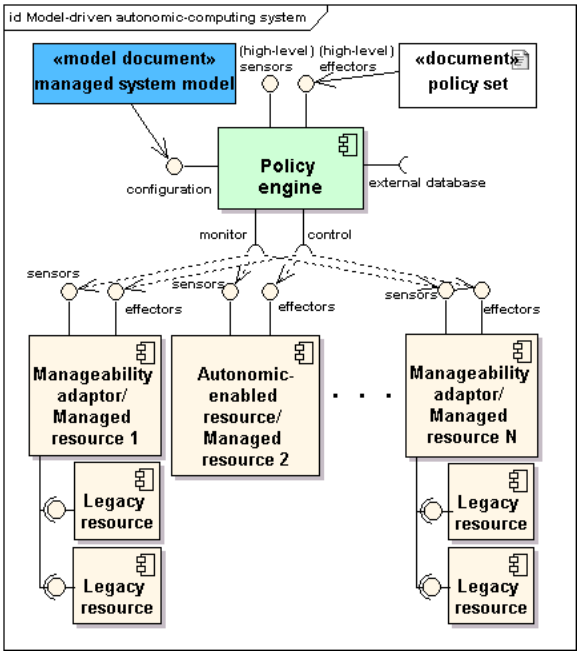

Figure 1. The autonomic architecture.

Manageability adaptors Adaptors are used to expose the manageability of sets of legacy resources in a uniform way, through *sensors* that allow access to resource state and *effectors* that enable the configuration of resource parameters in line with policies set up by the system administrator. The Management Using Web Services (MUWS) standard [6] is the recommended approach to implementing these adaptors, as it defines a resource management architecture that leverages web service technology benefits such as platform independence, loose coupling and security support.

Policy engine The universal policy engine has a *configuration* interface used to set up the engine for the management of a particular set of resources. The system model supplied using this interface specifies the relevant system resources,

Fourth International Conference on Autonomic Computing (ICAC'07)
0-7695-2779-5/07 $20.00 © 2007 IEEE

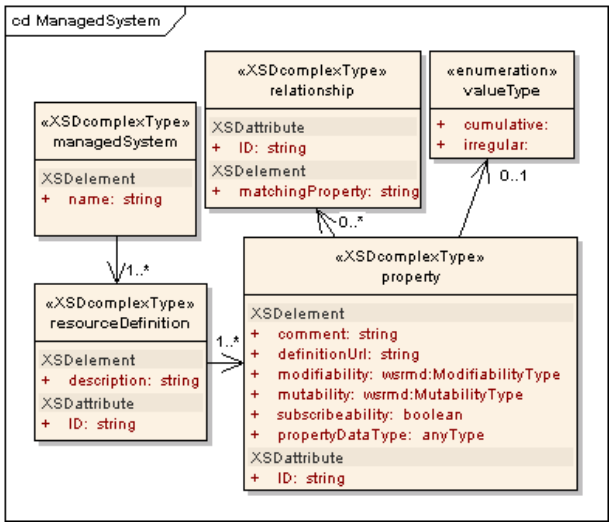

Figure 2. Managed system metamodel

thus also defining the policies that the engine will accept through its *(high-level) effectors* interface. The engine implements a policy set by *monitoring* and *controlling* the sensors and effectors of the managed resources, respectively. The "high level" resources of the managed system are exposed through the *(high-level) sensors* interface, enabling the inclusion of the system into a larger one, a key requirement for the design of manageable systems of systems [1].

Internally, the engine comprises a module for evaluating the expressions in policy scope definitions, business value specifications, conditions and actions; an internal clock for time-based expressions; and a policy resolution and scheduler. An internal cache can optionally be used in addition to an external database for storing state information. The engine can be implemented as a standalone software application/service, as a web application running within an application server, or even as a hardware appliance.

Managed system blueprint The model or *blueprint* used to configure the policy engine is an instance of the managed system metamodel in Figure 2. A managed system is a named sequence of resource definitions (*resourceDefinition*) comprising unique identifiers, descriptions and sets of resource properties with their characteristics. Each resource property is strongly typed (*propertyDataType*) and is assigned a unique ID and the URL within a metadata repository where its definition is available.

Policies A combination of policies (Figure 3) tell the policy engine how to manage the underlying system. *Resource definition policies* specify the higher-level resources exposed by the system, e.g., to report on its state or to enable its integration into a larger managed system. *Resource configuration policies* specify the desired values of modifiable resource properties as functions of the state of the managed system and of time. *Resource scheduling policies* describe

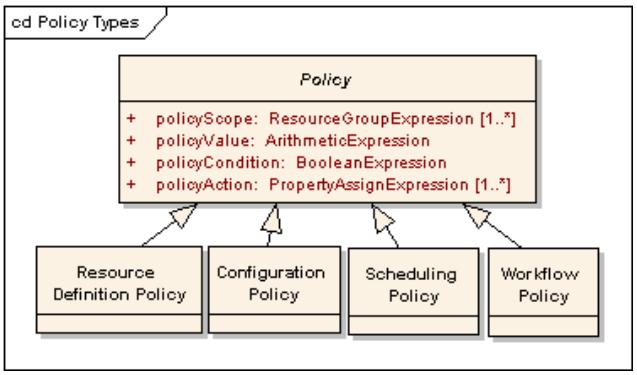

Figure 3. Supported policies

how system capabilities specified by certain resource properties (e.g., server CPU) is to be allocated within the system. Finally, *workflow policies* are used to declare sequences of actions that the policy engine is required to execute.

3. Conclusions

Based on insights gained from the development of a commercial framework for policy-driven resource allocation [2], we introduced a model-driven autonomic architecture built around a universal policy engine capable of managing heterogeneous combinations of legacy and autonomic-enabled resources. Work is underway to enhance the flexibility of the supported policy types, and to validate the use of policies in exposing the "high-level" resources of the managed system, so that it can be integrated as a managed resource into a system of systems [1]. Future work will include the development of an IT metadata repository from which the system blueprints used by the architecture will draw their resource definitions, thus enabling the design of reusable policies and policy templates.

References

[1] Y. Bar-Yam et al. The characteristics and emerging behaviors of system-of-systems. NECSI Report, January 2004. http://necsi.org/education/oneweek/winter05/NECSISoS.pdf.

[2] R. Calinescu and J. M. D. Hill. System providing methodology for policy-based resource allocation, July 2004. United States Patent Application no. 10/710322.

[3] IBM. Policy Management for Autonomic Computing. http://dl.alphaworks.ibm.com/technologies/pmac/PMAC12_sdd.pdf.

[4] J. Kephart and D. Chess. The vision of autonomic computing. *IEEE Computer Journal*, 36(1):41–50, January 2003.

[5] B. Moore. Policy Core Information Model extensions, 2003. IETF RFC 3460, http://www.ietf.org/rfc/rfc3460.txt.

[6] B. Murray et al. Web Services Distributed Management: MUWS primer. OASIS WSDM Committee Draft, 2006.

[7] Sun Microsystems, Inc. Sun^{TM} Grid Compute Utility, 2006. http://www.sun.com/service/sungrid/SunGridUG.pdf.