Policy and Profile: Enabling Self-knowledge for Autonomic Systems

Ray Carroll[1], John Strassner[2], Greg Cox[2], and Sven van der Meer[1]

[1] TSSG, Waterford Institute of Technology, Waterford, Ireland
{rcarroll, vdmeer}@tssg.org
[2] Motorola Labs, Schaumburg, IL, USA
{john.strassner, greg.cox}@motorola.com

Abstract. The standard definition of autonomics is that of self-governance, including such properties as self-configuring, self-healing and self-optimizing. To really enable self-anything, however, we must first deliver another 'self-' property - self-knowledge. We define self-knowledge as information about a system enabling it to reason on its own capabilities and actions. This knowledge can come in many forms but we propose that there are essentially two key elements: knowledge of the individual parts of the system, and knowledge about the rules that determine the interaction of these system components. This paper presents a model describing self-knowledge, with policy for defining rules and profiles to express the individual entities knowledge.

1 Introduction

Autonomic management [1] aims to ease system management by abstracting complexity and making common management tasks the responsibility of the system, rather than the administrator. A critical objective of autonomic systems is the need to be able to adapt to changes in the managed environment. For example, an autonomic *network* [2] would adapt it's services and resources in accordance with changing environmental conditions and user needs. In order for any system to be capable of this sort of autonomic management, it must be able to understand its own component parts and the combined effect of these parts. In essence, autonomics is about self-knowledge [3] and the usage of this knowledge to determine an appropriate action. It is our supposition that self-knowledge is essentially a function of two factors.

1. Knowledge of the rules that govern the system: high-level business rules which determine the overall goal of the system and specific rules (scoped by higher-level rules) that govern the interaction of system parts for specific situations.
2. Knowledge of the system itself, i.e. all relevant managed elements within the system and all their relevant possible roles/functions and data.

In this paper we present a model describing the factors listed above in the form of policy (1), profile and roles (2) and the relationships between these. Policies are rules that govern a system and its components, and hence represent information about how the system acts. The combination of profile and roles provide individual entity data

R. State et al. (Eds.): DSOM 2006, LNCS 4269, pp. 239 – 245, 2006.

and behavioural information. Section 2 describes our policy framework and section 3 explains our profile framework. Section 4 then describes our model of how policy, profile and roles interact to facilitate self-knowledge for autonomic network management systems. Section 5 then presents our conclusions and future work.

2 Policy

As mentioned in the introduction section, policy rules are used to govern an autonomic system. In the DEN-ng [4] model policy is realized as an Event, Condition, Action (ECA) triplet, having the semantics: "ON event, evaluate condition clause, THEN execute appropriate actions in the action clause". One or more of a PolicyRule's PolicyEvents trigger the evaluation of a PolicyRule. A PolicyEvent may contain flexibly defined combinations of events (PolicyEventComposite) or individual events (PolicyEventAtomic). As such, a PolicyRule may be triggered on a combination or sequence of events.

When a PolicyRule is triggered by PolicyEvents, evaluation of a PolicyRule's PolicyConditions occurs. Similar to PolicyEvents, PolicyConditions also can include combinations of conditions as PolicyConditionComposite or individual conditions as PolicyConditionAtomic. Further, an attribute of PolicyConditionComposite allows for specification of whether the composite condition is expressed in Conjunctive Normal Form or Disjunctive Normal Form. These features and others in the DEN-ng policy model allow for the expression of complex PolicyConditions in a PolicyRule.

When a PolicyEvent triggers evaluation of the PolicyConditions, then one or more of a PolicyRule's PolicyActions can occur. There are two types of actions: pass actions are invoked if the condition is TRUE, and fail actions are invoked if the condition is FALSE. Like before PolicyActions can include combinations of PolicyActionComposite objects and/or PolicyActionAtomic objects. As such, complex actions and sequences of actions can be expressed using the DEN-ng policy model. Taken together, all these features provide a rich expression of policy needed to enable knowledge of the rules that govern the interaction of system parts in today's complex systems. The need to accommodate multiple views as described in the DEN-ng Policy Continuum [4] further motivates this rich means of modelling policy, acknowledging that the various stakeholders in a complex system have different views of policy and degrees of abstraction in policy expression. DEN-ng defines 5 views: Business, System, Network, Device and Instance. Policies become more specific and increase in technical detail as one moves from the Business to the Instance View. Other policy models exist in the art (e.g., Ponder policy specification language) and are discussed in [4]. However, these alternative policy models tend to be less general than the DEN-ng ECA approach, often tailored to a specific use. As such, this work focuses on the DEN-ng approach employing ECA rules.

3 Roles and Profiles

We initially listed two items that are important for self-knowledge in any system, where Policy provides the first of these. The second was knowledge about the individual components of the system. Our approach to modelling this information is to develop a

generic framework that allows us to define entity information in a standard yet flexible way. The principle aim is to allow entities have different sets of information as per their functionality and also to allow information from various sources be associated to an entity. As such we propose the concepts of roles and profiles.

3.1 Roles

Typically, the behaviour of a managed entity is represented by the actual methods and properties of that entity. However, methods and properties alone are not enough to enable self-management as they do not allow us to fully understand an entities function or how it interacts with other entities. In many cases, an entity may have different attributes, tasks or functions that depend on the current situation, and so will use different sets of attributes and methods to execute a task. This enables the entity to adapt to a particular context. The DEN-ng model uses the role-object pattern [5] and we introduce the idea of **Role** to abstract this, so that the different situational requirements are not dependent on individual entities. This enables the model of the entity to be separated from the model of the functions that the entity takes on, and is seen in Figures 1 & 2. Here we see that an Entity (e.g. Organisation or Individual) can aggregate zero or more EntityRoles. This enables the Entity to take on the characteristics of two different entities (e.g. Vendor and Service Provider) without changing the attributes of the Entity directly. This reflects the real world as the Entity itself did not change, only the role that it was playing at a given time.

Role is a well established concept in terms of access control [6] and has also received attention in context–aware and ubiquitous systems [7, 8]. For us, role presents a means to specify what information is relevant based on the current function it is trying to fulfil. Thus, roles present an extensible means for us to introduce contextual characteristics and behaviours, modelled as classes, into the model. Roles also allow separate sources of data that together prove more useful to be grouped together. Fig. 1 shows that EntityRole has an aggregation relationship to Data. This enables data to be associated to an entity via its roles, making pertinent information available based on the set of active roles of an entity.

3.2 Profiles

There are many initiatives that propose the use of profiles at some level. However, most of these are focused on some very specific purpose. Our aim is not to specify particular profile data models, but rather to develop a profiling framework for integrating these disparate sources of information. We believe this will not only increase system knowledge, but also improve accessibility, reusability and overall usefulness of an entity's information. We define a new variant of the role object pattern [5], where the Role object itself aggregates new information. We call this new information Data and use the composite pattern to define appropriate subclasses of data. This enables additional data to be associated to a particular role, instead of embedding the data in a role. We then specialise data to suit the needs of a Profile (Figure 1). This enables each set of profile information to be defined independently and attached to a role at runtime. A Profile is a container for data and we specialise this to EntityProfile, for data about a specific entity. We then use a Composite Pattern [9] to separate EntityProfile into a container (EntityProfileComposite) and component

(EntityProfileComponent) enable hierarchies of Profiles to be built. This allows us to provide a single profile that can contain many profile components and enhances the approach specified in initiatives like 3GPP Generic User Profile [10].

An EntityProfileComponent is also a container for ProfileData. ProfileData is subclassed from EntityData, which is a generic class for any data related to some entity. This structure provides future flexibility and extensibility for our design. ProfileData is the actual data about an entity that composes a Profile. Again, the composite pattern is used so that ProfileData can be either atomic or composed of other data (i.e. nested data values). Figure 2 gives a simple example of a user entity and their associated roles and profiles.

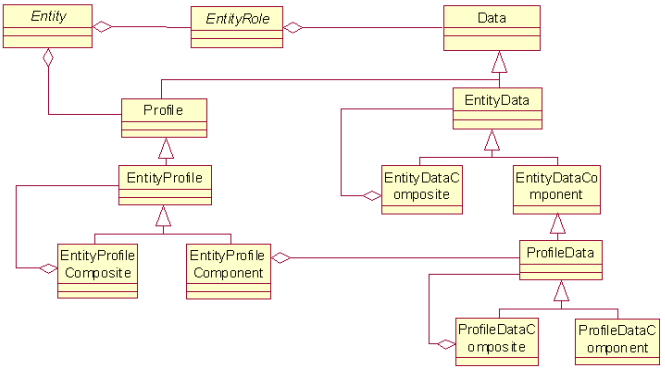

Fig. 1. The Data model

The entity JoeBloggs has a role of Employee and a profile jbloggsProfile which aggregates EntityProfileComponents (LIP[11] and CC/PP[12]) and provides management functionality such as adding, removing, updating, querying etc. It is also important that an entity (e.g. JoeBloggs) is associated to its data (e.g. jbloggsProfile) directly (not just via Roles) for more integrated management of profiles. For Joe Bloggs' role of Employee two profile components are relevant (i.e. LIP and CC/PP). This example is based on a user scenario, but the same mechanism may apply to any entity, (e.g. services and resources). This reflects the needs of autonomic systems, which require functionality to change with changing user needs and/or environmental conditions. In our system, we meet this challenge by dynamically instantiating new roles using *policy*. The combination of roles and policy then enable which *profiles* are allowed to be used.

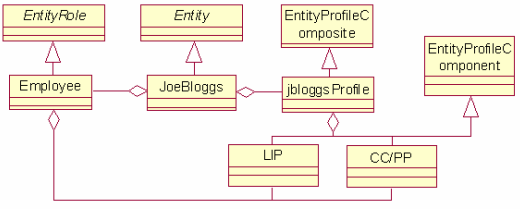

Fig. 2. Example of Entity JoeBloggs with role Employee, profile and profile components

4 Model for Enabling Self-knowledge

In the previous sections we described the individual parts of our proposed system that are core to enabling self-knowledge. However, these parts are not sufficient on their own; rather it is the overall view of how they interact (Figure 3) that makes self-knowledge possible. As the approach to autonomic management that we have adopted is policy-based, policy is a core part of the system and, in effect, governs our system at every level. While each entity provides its own behaviour, policy determines what parts, under what circumstances and to what effect that behaviour is executed. As we can see from Figure 3, Policy **governs** Entity, EntityRole and Data. As regards Entity, Policy will govern specifically their creation, management, and deletion, as well as the addition and removal of roles (and hence profiles) associated with an Entity. Specific semantics of different management functions controlled by Policy for an Entity are defined by the PolicyEntityDetail association class.

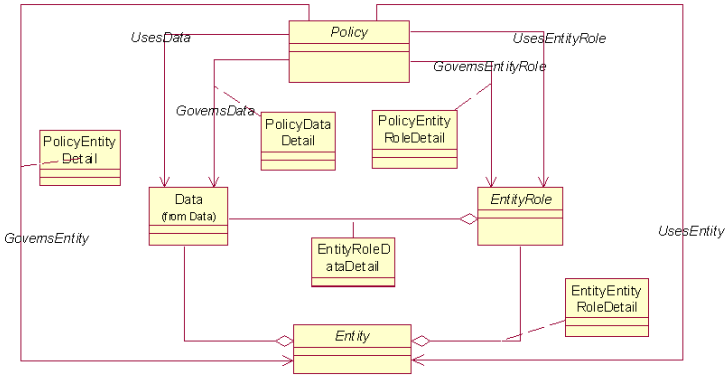

Fig. 3. Overall model of policy, data, entity and role (subclasses omitted for readability)

Since an Entity is largely characterised by its roles, the use of Policy to govern which EntityRoles an Entity can have provides both an abstract view of the Entity's functionality as well as detailed control over the Entitys characteristics and behaviour. Based on the Policies governing the system and knowledge of the environment and users, the EntityRoles relevant to the Entity in question must be determined and enabled or disabled accordingly. This in turn controls the functionality that an Entity has at any given time. The particular semantics of which EntityRole a particular Policy can select is given in the PolicyEntityRoleDetail association class. Multiple EntityRoles can be selected by the same Policy for different reasons and enabling policies to be reused but tailored to the specific needs of a given EntityRole. For example, a device may have many subclasses of the same role (e.g. EdgeDevice, DSLEdgeDevice) but implement that role using very different functionality (e.g. PC v Switch v Router). The Policy should stay the same ("give access") but the particular mechanisms used will vary. This association class enables these details to be captured while keeping the same abstracted pattern. The relationship of EntityRole to Data is

characterized by the EntityRoleDataDetails class. This enables specific semantics to be attached to how a given EntityRole uses particular Data.

The third and final governance relationship Policy has is with the Data class. This relationship signifies that Policies will also govern Data; by this we specifically mean that Policy will govern the set of Profiles that an Entity can have. Note, however, that Data is aggregated by EntityRole. In effect, Policy will determine the set of EntityRoles that an Entity can have; based on this, the set of Profiles that are allowed to be used is subsequently determined. As before, the PolicyDataDetail association class enables specific semantics for a given {Data, Policy} combination to be realized. *Policy* also has a number of *Uses* relationships with EntityRole, Data and Entity. These signify that policy will also use these classes in its inherent decision making process.

5 Conclusions and Future Work

This paper has presented a novel design for capturing self-knowledge in autonomic systems using the DEN-ng information model and policy design. In addition, a novel combination of policy, profile and role interaction has been presented: in our system, *Policy* is used to enable or disable the set of *EntityRoles* that a given Entity has; the combination of EntityRole and Policy is in turn used to enable or disable the set of *Profiles* that a given Entity can use. This enables system changes to trigger PolicyEvents, and those PolicyEvents to control the functionality of an Entity (via its roles and Profiles). This enables the autonomic system to (indirectly) use Profile information to control the resources and services that a network provides, as well as those that a user can utilise, as a function of context. Furthermore, it is an extension of the Shared Information/Data model, which is standardised in the TeleManagement Forum (also being considered for standardisation in ETSI TISPAN and ITU-T).

While we have described a model that we feel can enable self-knowledge, we have not yet addressed the issue of how this model can be utilised in a real system to provide autonomic management. A model is a view of the systems underlying data structure and does not provide system behaviour. As such future work will investigate algorithms for processing the components of self-knowledge to truly produce autonomic network behaviour and in line with this also investigate the relationship of this work to context.

References

[1] Kephart, J.O. and Chess, D.M., “The Vision of Autonomic Computing”, IEEE Computer, January 2003, www.research.ibm.com/autonomic/research/papers/

[2] Strassner, J. and Kephart, J., “Autonomic Networks and Systems: Theory and Practice”, NOMS 2006 Tutorial, April 2006

[3] Thomas Cofino et al "Towards Knowledge Management In Autonomic Systems," *iscc*, p. 789, Eighth IEEE Symposium on Computers and Communications, 2003.

[4] Strassner, J., “Policy-based Network Management: Solutions for the Next Generation”, Morgan-Kaufman Publishers. ISBN 1-55860-859-1, (2004)

[5] The Role Object (Design) Pattern. Download PDF from http://www.riehle.org/computer-science-research/1997/plop-1997-role-object.pdf
[6] National Institute of Standards and Technology (NIST), Role Based Access Control. http://csrc.nist.gov/rbac/
[7] Beresnevichiene, Y**.** "A Role and Context Based Security Model", Technical Report, University of Cambridge, Computer Laboratory January 2003.
[8] Sashima, A. et al. 2006. Toward Role-based Agent Coordination for Mobile and Ubiquitous Services. In *Proceedings of the 20th international Conference on Advanced information Networking and Applications, Volume 02* (April, 2006). AINA. IEEE Computer Society, Washington DC, 262-266.
[9] E. Gamma, et. al., "Design Patterns", pp 163-173
[10] 3GPP, Generic User Profile, http://www.3gpp.org.
[11] IMS Learner Information Package, http://www.imsglobal.org
[12] W3C "CC/PP: Structure and Vocabularies", 2004. http://www.w3c.org