Policy-based management of middleware for distributed sensor applications

Nelson Matthys, Sam Michiels, Wouter Joosen, and Pierre Verbaeten
IBBT-DistriNet, Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
nelson.matthys@cs.kuleuven.be

ABSTRACT

Contemporary distributed software systems have become extremely heterogeneous, dynamic and large-scale. They include back-end servers, regular PCs and various mobile and embedded devices, as well as diverse network infrastructures, such as sensor networks. However, these factors make manual management of the distributed system and its installed software extremely complex. In this position paper we seek for an architectural solution for a middleware that supports autonomic management of these distributed software systems through a policy-based approach. This middleware architecture suggests the introduction of a closed control loop on top of a highly customizable middleware architecture. We describe the key research challenges that still remain open for discussion and propose a plan to tackle them.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems, Distributed Applications

Keywords

middleware, software architecture, policy-based management

1. PROBLEM CONTEXT

As the environments in which distributed software applications execute become more dynamic, heterogeneous, and integrated with the physical context, a new class of heterogeneous distributed software systems has emerged. These software systems are composed of geographically dispersed systems using a variety of devices and platforms to deliver services that cross service, platform and system administration boundaries.

Applications in the context of supply chain logistics serve as representative examples of such distributed software systems. A warehouse management application for a logistics enterprise, e.g. relies on a heterogeneous combination of

devices and networks: (1) a fixed network of powerful enterprise resource planning servers in the back-end, (2) wireless gateways such as WiFi hotspots distributed over multiple warehouses, trucks, and trailers, and (3) many thousands of various kinds of programmable sensor nodes attached to products or containers for collecting data on temperature, humidity, or location. Users can then retrieve these values in order to get a full trace of their products during transport.

However, using and managing these distributed software systems is not straightforward. First of all, the heterogeneity of the infrastructure, mainly specified by the application domain and its requirements, requires sufficient knowledge of the developer, user, or manager. Secondly, achieving quality related requirements like reliability, efficiency, or availability, especially given the scale of deployment of the software system is not easy. Effectively using these large-scale distributed systems therefore requires an approach that enables them to be managed and configured easily and uniformly through, for instance, a policy-based approach. The logistics application case, for example, clearly motivates the need for such an approach. The application may, e.g. specify a policy that expresses that the most efficient services in the environment must be used (that still provide an acceptable quality level). This means, e.g. that during the products' transport, the localization service installed on the truck is most suited to trace the transported goods since it is less resource and energy-constrained. Also there is only a limited loss of quality in the location data, since the container's location will always be very close to that of its carrier. Only if the container is stored outside, then the more accurate, but less energy efficient localization service installed at the container is to be used.

Such a policy-based management approach consists of several steps, which form some kind of closed loop: information gathering about the execution context, interpretation of policy files, matching their conditions and triggering any related events, and the enforcement of their associated actions on the current execution context. However there exist certain challenges in this approach, e.g. regarding the granularity of the artifacts to be monitored and adapted. But also, especially in the context of such complex distributed systems, conflicting policies may coexist (e.g. regarding different applications), and some devices may also be better suitable to be equipped with particular parts of this control loop (e.g. low-end sensor nodes vs. high-end back-end systems). Finally, especially given the scale of deployment and its various requirements, this policy-based management should happen autonomously, without involving a human in the loop.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MiNEMA'08, April 1, 2008 Glasgow, Scotland
Copyright 2008 ACM 978-1-60558-122-4/08/04 ...\$5.00.

The remainder of this paper is structured as follows: Section 2 gives an overview of a software architectural solution for a middleware, manageable by policies. Section 3 zooms in on some of the key challenges related to the approach, while Section 4 proposes a plan to tackle and evaluate them.

2. ARCHITECTURE

The typical approach to support the development, deployment, execution and maintenance of distributed applications is by using middleware. Middleware is system software that connects otherwise separate applications and platforms and offers generic facilities such as abstraction of system details through higher-level APIs, interoperability, reusable services, and a deployment and execution environment for distributed applications. Middleware offers programming abstractions to mask (some of) the heterogeneity of the underlying device-specific execution environment in individual network nodes.

2.1 Basic architecture

The basic software architecture, illustrated in Figure 1, for the middleware leverages on our approach in [3] and consists of several customizable component frameworks. With customizable frameworks, we mean that all component frameworks are highly configurable in the sense that due to the diversity and dynamism of the environment and application requirements, existing functionality can be tuned to the underlying host and the application scenario, and additional functionality can be deployed at runtime.

We identify four main customizable component frameworks in the middleware, each responsible for a particular aspect of the middleware. The *execution environment* is responsible for keeping track of the installed configuration (i.e. through a system graph), implementing the core methods for managing the component frameworks in a device-specific language, and device-specific tuning of the underlying execution environment (e.g. memory management and concurrency). Secondly, to hide the physical distribution of the environment to the other middleware levels and the application, we added a *distribution framework* which is comparable with the distribution layer in traditional middleware systems [5]. Thirdly, common middleware functionality, e.g. services responsible for data aggregation, persistence, or advanced group communication, is included in the *customizable service frameworks*. Finally, the *adaptation architecture*, structured around the entire middleware, is responsible for managing the entire middleware configuration (i.e. gathering information about the entire system, achieving safe adaptations). This adaptation architecture forms the basis for the control loop, which will be explained in the following section.

On top of this middleware resides the application software. In the example sketched in the previous section, representative application software for logistics includes, for example, a temperature sampling service in the sensor network, a localization service on the gateway or truck, and a warehouse management system that analyzes all incoming data of the products and constructs a full trace of them (see Figure 2).

2.2 Control loop

The control loop is added on top of this architecture and is designed to enable policy-based management of the distributed system. In Figure 1, the loop is an integral part of the adaptation architecture and is illustrated in gray.

Figure 1: Overview of a architecture for a configurable middleware. The control loop is part of the adaptation architecture and is illustrated in gray.

The loop itself consists of three main parts: a monitoring framework, a policy management framework, and a reconfiguration framework. The *monitoring framework* captures valuable information about contextual properties. For example, the monitoring framework should be able to detect changes in the environment (e.g. availability of a nearby more efficient localization service on a truck or ship), and it should detect overload situations before they actually lead the run-time system to crash. The monitoring framework may gather this information by using probes on resource-constrained devices for measuring system states, but it may also try to push management and intelligence deeper into the network.

The monitoring framework passes this information to the *policy management framework* that can interpret and enforce the policies that were injected in the distributed system. As already stated, these policies may reflect preferences regarding trade-offs to be made when deploying middleware services in the network. A typical trade-off, as described in Section 1, is the choice between a highly efficient but inaccurate service and a highly precise, but resource-intensive alternative. For example, using the localization service installed on a ship's powerful node provides less accuracy (QoS), but is more resource-efficient overall, than installing the service on each of the carried containers. Additional policies in the distributed system can then also express that whenever the ship's localization service takes over, all container nodes may go into sleep mode to conserve power.

Finally, the outcome of the policy-reflection process is then executed by the *reconfiguration framework*. The reconfiguration engine in the framework will decide whether the action can be executed or not (e.g. when the load is below a certain threshold), and will prepare for the actual reconfiguration. The reconfiguration engine may have to collaborate with neighboring nodes in order to update the network in a consistent and safe manner. For example, before a service can be removed, all parties that are using it must be informed about the reconfiguration that will take place.

2.3 Deployment view of the architecture

A typical deployment view of this middleware is illustrated in Figure 2. In this figure, the control loop is distributed across several tiers. Inside the sensor network tier, the mon-

Figure 2: Deployment view of the middleware. Each of the tiers is customizable according to the different requirements and capabilities. The control loop is distributed in a decentralized manner across these tiers.

itoring framework consists of probes that gather contextual information and pass this information to the policy management framework on the gateway. The gateway can, based on the received information, interpret and enforce the associated policies itself, or pass it towards the policy management framework on the enterprise tier, where it can be matched against other business policies (e.g. high availability). Both policy frameworks can then decide together on the most optimal outcome, which can be enforced in all involved tiers.

3. RESEARCH CHALLENGES

This section states the remaining key research challenges and objectives that have to be solved or taken into account:

- Is it feasible that the control loop is distributed in a decentralized manner across various parts of the infrastructure? Does each node in the distributed system need to have an similar instantiation of this control loop, or can the infrastructure be equipped with probes and can the interpretation and enforcement of policies happen on more efficient devices (see Figure 2). What is the trade-off in efficiency between a fully decentralized control loop or a centralized control loop?
- How to manage competing and coexisting policies (e.g. from different applications) and reconfiguration rules in the middleware system? How to find the optimal solution without affecting the entire distributed system (trade-off in efficiency of possible configurations)?
- What are the system artifacts that need to be monitored to enable realistic policies and reconfiguration rules. What features for monitoring should the corresponding component model offer? How fine-grained can this monitoring be? Is this monitoring a periodic or continuous process (i.e. in order to achieve complete autonomous policy-based management)?
- What are the system artifacts that can be adapted by reconfiguration rules and policies? How fine-grained should this adaption be? What features for adaptation should the corresponding component model offer?
- How to define domain-specific policy languages that are easy to use and express concerns by a wide range of users (developer, network manager, application end-user). To what extent can the system's behavior be programmed as policies. How to measure this expressiveness and ease of use?

- What existing software technologies can be used? Reinventing the wheel by defining a new component model and middleware is not realistic because of the small research community one can only address.

4. RESEARCH PLAN

First of all, we will focus on a fixed set of scenarios in our logistics' case where the proposed policy-based management approach seems feasible. This will deliver us specific requirements that define what artifacts should be monitored, how regularly they should be monitored, and what reconfiguration rules should be defined for each particular scenario. It should also allow us to define an appropriate domain-specific policy language. Secondly, we are aware that this policy-based approach may greatly benefit from existing research on adaptability in open component-based middleware [2], reflection [1], and autonomous behavior [4]. Thirdly, since our goal is to address a large community, existing technologies like (nesC, OSGi, Java EE) will be used to develop the proposed approach. Finally, once a prototype of this middleware has been developed, it will allow us to differentiate between configurations. This means that we can evaluate the trade-offs of efficiency (in resource consumption) between deployment scenarios (centralized vs. decentralized control loop) and software configurations (current running configuration vs. newly proposed configuration).

5. REFERENCES

- [1] F. Kon, M. Román, P. Liu, et al. Monitoring, security, and dynamic configuration with the dynamictao reflective orb. In *Middleware, LNCS*, volume 1795, pages 121–143, 2000.
- [2] G. Coulson, P. Grace, et al. A component-based middleware framework for configurable and reconfigurable grid computing. *Conc. and Comp.: Practice and Experience*, 18(8):865–874, July 2006.
- [3] N. Matthys, S. Michiels, W. Joosen, and P. Verbaeten. End-to-end middleware for distributed sensor applications. In *Proceedings of the 5th MiNEMA Workshop*, pages 68–72, September 2007.
- [4] P. Oreizy, M. Gorlick, et al. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.
- [5] D. C. Schmidt. Middleware for real-time and embedded systems. *Communications of the ACM*, 45(6):43–48, 2002.