Middleware Support for Embedded Software with Multiple QoS Properties for Ubiquitous Computing Environments

Stephen S. Yau, Yu Wang and Dazhi Huang
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406, USA
{yau, wangyu, dazhi.huang}@asu.edu

Abstract

Ubiquitous application software usually has multiple QoS requirements, such as situation-awareness, real-time, and security, which make the application software development complicated. In this paper, an approach to supporting multiple QoS properties in application software using middleware is presented. Our Reconfigurable Context-Sensitive Middleware (RCSM), which provides situation-awareness support to the application software, is expanded to support more QoS by using Aspect-Oriented Software Development techniques. In the expanded RCSM, each QoS is processed through a set of aspect components. The application developers specify the QoS requirements in a specification file, which is in turn compiled to configure the corresponding aspect components. Being associated with certain application objects, the QoS aspect components enforce the QoS requirements according to the specification during run-time. In this paper, security, in addition to situation-awareness, is used as an example to illustrate our approach.

Keywords: *Ubiquitous computing, embedded software, Reconfigurable Context-Sensitive Middleware, QoS, Aspect-Oriented Software Development, situation-awareness, security.*

1. Introduction

The vision of ubiquitous computing (ubicomp) [1] is part of the stimulation for the emergence of the portable computing devices, such as PDA, pocket PC, and tablet PC. The application software residing on these devices often have some QoS requirements or constraints, such as situation-awareness, security, and real-time performance. S*ituation* is a set of past contexts and/or actions of individual devices relevant to future device actions [2]. *Context* is any instantaneous, detectable, and relevant condition of the environment or the device, such as time, location, light-intensity, noise-level, and available bandwidth. *Situation-awareness* is the capability of monitoring the context, detecting situation changes and responding to the changes.

These QoS requirements make the design and development of ubicomp applications complicated. The ubicomp systems must have a combination of the following capabilities: monitor the temporal, spatial and physical conditions of environment, adapt to the environment, monitor or even control the continuous dynamics of the system, satisfy real-time requirements, and enforce security constraints.

Middleware has been widely used to support the application-specific QoS properties in enterprise networks. By separating the QoS-related part from the application software and move it to the middleware layer, the burden of developing the QoS-related software is alleviated for the application developer. The middleware approach has the following three advantages: (1) The development of application QoS properties is more efficient and error-proof since application developers switch from a monolithic ad-hoc development to a layered systematic development using middleware services. (2) Maintenance of QoS aspects of application software is easier due to its development based on uniform middleware service. (3) Resources can be more efficiently shared by multiple applications residing on one device through middleware.

2. Current State of the Art

Although middleware has been used to support QoS for general network applications, the current results are insufficient for ubicomp applications due to the following two shortcomings:

First, middleware is rarely designed for wireless networks. Most middleware systems are for enterprise networks, such as TAO [3] and QuO [4]. TAO [3] focuses on the real-time aspect and the real-time requirements in TAO are represented by pre-defined IDL interfaces and enforced based on priority queue in ORB. QuO [4] provides a toolkit, which includes a suite of quality description languages (QDL), compilers, and library components to ease the burden of QoS programming. These two are not suitable for ubicomp applications in wireless networks.

Second, most middleware systems deal with the QoS attributes such as respond time and throughput. But no QoS properties, such as situation-awareness and security are included in these middleware systems. Besides Tao [3] and Quo [4], these systems include Agilos [5] and Q-RAM [6]. Agilos [5] focuses on how to control QoS adaptation in middleware architecture, and the QoS is specified by fuzzy rules and membership functions. Q-RAM [6] deals with the QoS management problems using a resource allocation model, and the QoS is represented by resource utility functions.

Our current Reconfigurable Context-Sensitive Middleware (RCSM) [2,7,8] is capable of supporting one QoS, situation-awareness, for application in wireless networks environments. RCSM is a middleware for ubicomp environments, and its architecture is shown in Figure 1. RCSM provides middleware support for:

- *Situation-awareness:* RCSM is capable of responding to situation changes by activating appropriate actions.

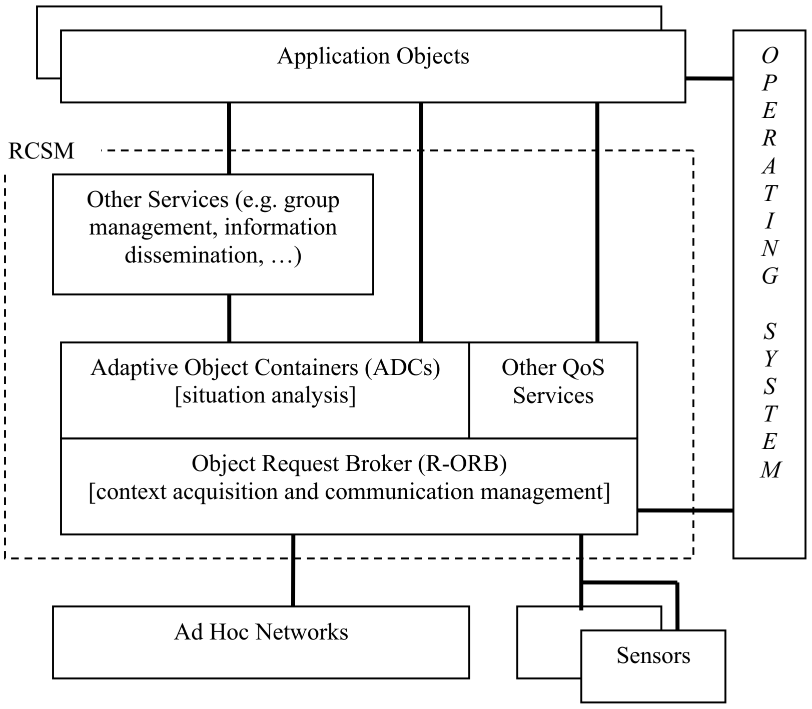
Application Objects
RCSM
Other Services (e.g. group management, information dissemination, …)
Adaptive Object Containers (ADCs) [situation analysis]
Other QoS Services
Object Request Broker (R-ORB) [context acquisition and communication management]
OPERATING SYSTEM
Ad Hoc Networks
Sensors

Figure 1. RCSM architecture.

- *Ephemeral group management:* This service is under development. RCSM uses situation changes of a device to manage group collaborations between the host device and its group devices.
- *Autonomous coordination for information dissemination:* This service is under development. RCSM provides information dissemination service for situation-aware coordination among devices.

Situation-awareness is considered as a new type of QoS because it represents the capability of application software to understand and adapt to the situation changes. With the middleware support provided by RCSM, application developers can develop application software with this new QoS requirement (situation-awareness) more easily and systematically. A test bed, Smart Classroom [2], is being constructed to demonstrate our approach in supporting situation-awareness of application software using RCSM.

3. Situation-awareness in RCSM

Let us first show how RCSM supports the development of the software with the situation-awareness requirement.

The difficulties for an applications having situation-awareness property are in context acquisition and situation processing. A straightforward way of building a situation-aware application is to put everything together: context acquisition, situation processing, and utilization of the situation in the application. All these functionalities are mixed together and the application developers have to incorporate all these aspects in the application software development, including the non-traditional input acquisition from sensors and the processing of raw context data. These difficulties make the development of situation-aware application software a big challenge. Moreover, application software developed using this approach will be difficult to maintain or reuse due to its complexity.

Our middleware approach is to separate context acquisition and situation processing from situation-utilization. We move the context acquisition and situation processing into the middleware layer so that application developers can focus on the application development. Due to the application-specific nature of the situation-awareness requirement, the situation-processing component in the RCSM is also application-specific. To facilitate application developers to specify situation-awareness requirements, we have developed a Situation-Aware Interface Definition Language (SA-IDL) [2].

Using the SA-IDL compiler, we can automatically generate a situation-processing component, *Situation-Aware Adaptive object Container* (SA-ADC), corresponding to a specification file in SA-IDL. We then import this component into the RCSM to make it a customizable situation-aware middleware for processing the application-specific situations.

During runtime, raw context data is collected periodically by R-ORB [7] and propagated to SA-ADC for processing, SA-ADC in turn checks if the situation changes defined in the SA-IDL file occur. If a situation change occurs, SA-ADC invokes the appropriate application actions defined in the SA-IDL file and implemented in application objects.

4. Using Aspect-oriented Method for Developing Multiple QoS in RCSM

Although our current RCSM supports only one QoS – situation-awareness, its modular architecture, as shown in Figure 1, makes it suitable for expansion to incorporate other QoS properties in the module of "Other QoS Services", such as real-time and security. Each additional QoS property will be supported by an additional component in expanded RCSM. In this section, we will show how to use

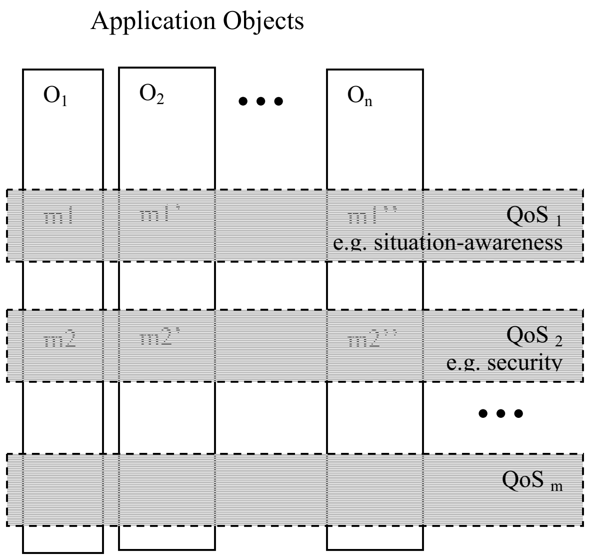

Figure 2. Application objects and crosscutting QoS properties.

Aspect-Oriented Software Development (AOSD) [9-11] to support multiple QoS in our middleware.

Expansion of RCSM to support additional QoS properties consists following three steps:

Step 1) Model QoS as a property that cut across the application objects residing on RCSM;

Step 2) Based on the QoS model, expand SA-IDL to specify multiple QoS constrains;

Step 3) Generate QoS aspect components according to the QoS specification, which provide runtime QoS support.

The first step is to model QoS. As shown in Figure 2, the application software is implemented as application objects (O1, O2, …, On) that reside on RCSM. Each object has its own methods, for instance, O1 has methods m1, m2, etc. A method may have some QoS constraints, such as situation-awareness constraint that specifies under which situation this method is invoked, and security constraint that specifies the security condition that should be satisfied before this method is executed. Methods of different application objects could have common QoS properties. For instance, method m2, m2' and m2'' have the same security constraint (see Figure 2). These common QoS constraints should be model as one aspect. Thus we model one QoS as an aspect property of application objects that cut crossing these objects, as depicted in Figure 2. Specifically, a QoS constraint is associated with a method of an application object. Using AOSD, we can efficiently organize the QoS properties in a uniform framework, or else the properties will be scattered in each application object and handled in a scattered and distributed way. The Separation of Concerns (SoC) discipline is used to organize these crosscutting QoS properties, such as situation-awareness, security, and real-time. Thus, we can develop a QoS at a time, and finally compose an application that satisfies multiple QoS requirements by associating various QoS with the application objects.

```
SecurityConstraint{
  Type-1 name-1; //internal variables
  … …
  Type-n name-n;
  Condition
       [on-satisfying] handler-1
       [on-violation] handler-2
}Constraint_name

RealTimeConstraint {
  Type-1 name-1; //real-time attributes
  … …
  Type-m name-m;
}Constraint_name
```

a. QoS constraints

```
Object {
  Type-1 name-1; //object attributes
  … …
  Type-k name-k;
  Method-1(parameter-1, …, parameter-i)
    withSecurityConstraint (Constraint_name-1)
    withRealtimeConstraint (Constraint_name-2)
  …
  Method-j((parameter-1, …, parameter-r)
    withSecurityConstraint (Constraint_name-s)
    withRealtimeConstraint (Constraint_name-t)
}
```

b. Object methods with QoS constraints

Figure 3. SA-CSL format

In the second step, we expand the SA-IDL based on the QoS model to a language, called *Situation-Aware Contract Specification Language (SA-CSL),* to specify multiple QoS requirements. SA-CSL has syntax that specifies the parameters of multiple QoS properties. Examples of QoS parameters include situation changes, security conditions, security policies, real time parameters, etc, as depicted in Figure 3a. SA-CSL also specifies the association between the QoS constraints and the application object methods as shown in Figure 3b. A QoS property is specified as a set of QoS constraints. Each QoS constraint is specified separately, and then it is associated with an application object method.

The third step is to generate QoS aspect components to provide runtime QoS support according to the SA-CSL specification. To address this, we will develop an SA-CSL compiler. Based on the SA-CSL specification, the compiler either generates a set of in-house components, or associates third-party components with application objects. Either way, these QoS aspect components provide specified QoS support to application software through the methods of associated application objects during runtime.

This approach is suitable for supporting QoS whose constraints are enforced at the *method* level, i.e., a defined QoS constraint is bound with a *method* of an application object. For instance, situation change is associated with a method of an application object and this method will be invoked when this situation change is recognized. As another example, a security policy is defined as a constraint associated with all actions to which this policy is applicable.

It is noted that this approach cannot support QoS whose constraints are not enforceable at the method level because this approach requires the application developers to breakdown the QoS and associate them with the methods of the application objects. For instance, availability of an application service cannot be enforced at the method level, and hence it cannot be supported by our approach directly.

5. An Example

In this section, we will show how to expand RCSM to support security.

First, we model security aspect and identify the elements of a security requirement. In general, the elements of a security requirement should include entities, actions, security mechanisms and policies. Entities are subjects and objects that are involved in the security management. There are several types of entities, including subject entities (component, application, user, device, host, group, network domain, etc.), resource entities (file, data, message, CPU, memory, etc.), and security entities (key, certificate, credential, role, etc.). Entities have

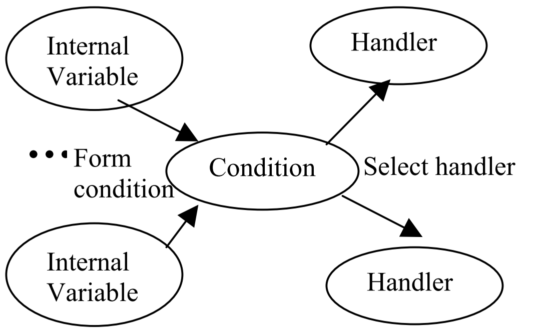

Figure 4 Structure of security constraint in SA-CSL

associated actions, which are taken by entities to interact with other entities, such as sending or receiving messages, uploading or downloading files, reading, writing, or executing files. Security mechanisms are basic security operations that constitute a specific security solution, such as encryption, decryption, re-keying method (for secure group communication), and digest method. Policies are high-level descriptions of the behavior of entities.

Second, we use SA-CSL to define these elements: Entities are defined as objects in SA-CSL; actions are defined as methods in objects; security mechanisms are either provided through system library or defined by developers; policies are defined as constraints (see Figures 3 and 4), which include internal variables, the condition formed by internal variables and handlers that enforce the security requirements based on the condition.

Figure 5 shows an example of SA-CSL specification for security. This example illustrates the following scenario: In a company, employees can be authorized by their managers and responsible security officers to access certain classified files through their PDA. Nobody is allowed to self-authorize access of a classified file and each authorization needs to be processed by both the responsible manager and the responsible security officer (separation of duty).

```
SecurityConstraint{
  //identity of operator
  string security_officer;
  string manager;
  //identity of target
  string subordinate;

  assert manager!=subordinate && security_officer
  != subordinate && manager!=security_officer;
  [on violation]
    ViolationHandler.report(security_officer,
    manager, subordinate, "Separation_Of_Duty");
} Separation_Of_Duty;

Object {
  …
  string identity(manager);
  //Before performing "authorize action"
  //Separation_Of_Duty constraint will be checked;
  //operator is an instance representing security
  //officer, target is an instance representing
  //subordinate
  authorize (File classified, Employee operator,
  Employee target);
  WithSecurityConstraint(new Separation_Of_Duty
  (operator.getIdentity(), identity, target.getIdentity()));
  …
} Employee;
```

Figure 5. An example of SA-CSL specification for security.

In the constraint specified in Figure 5, internal variables, security_officer, manager and subordinate, are checked in a condition expression. When the condition is violated, the handler, ViolationHandler, which is a component implemented by application developers, will report this violation. When we define the Employee object, this constraint is bound with authorize action.

Third, in the expanded RCSM, we use a Security Dispatcher (SecDisp) and a set of security handling components to support the security enforcement. The security handling components may include third party security components for encryption, authentication, etc. The technique that we use to design SecDisp is similar to Composition Filter [12]. After application developers finish the specification for an application, the compilation of this SA-CSL specification generates a table, which contains security-constrained actions, the condition specified in SecurityConstraint and corresponding handlers. The table is shown below:

Action	Condition	Handler_if_true	Handler_if_false
Employee. authorize	manager!= subordinate && security_officer != subordinate && …	Null. Simply continue the action	Violation Handler. Report
…	…	…	…

This table will be used by SecDisp to enforce security requirements in runtime. The runtime execution of SecDisp is depicted in Figure 6. Message m1 means there is a call to authorize method; m2 means the call to authorize method is allowed to proceed; and m3 is an invocation of report method in ViolationHandler. In runtime, if a message m1 is sent to SecDisp, SecDisp will evaluate the condition and make decision based on the result of evaluation. In the example, if the condition is satisfied, SecDisp will send m2 to allow the method call; otherwise, SecDisp will send m3 to invoke the report procedure in ViolationHandler.

This approach has the following two advantages: First, the security specification separates security

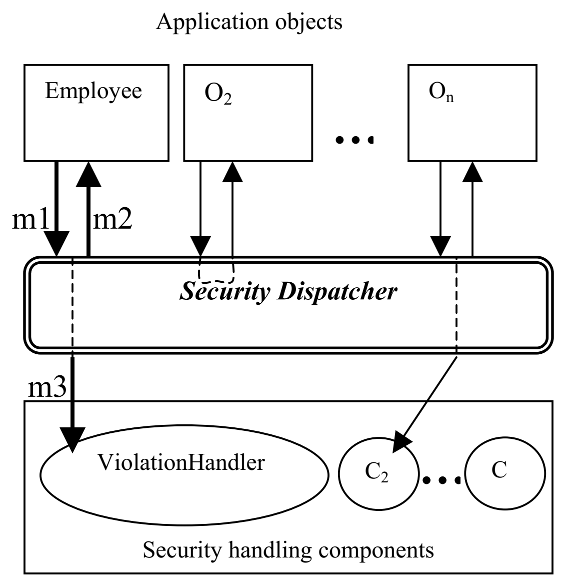

Figure 6. Security dispatcher.

constraints from application objects so that we can easily reuse the specification by binding security constraints with actions in other application objects. Secondly, in an SA-CSL specification, situation expressions and security constraints are both bound with actions defined in the entities so that situation-awareness and security can be supported simultaneously.

6. Discussion

In this paper, we have discussed how to use middleware to support multiple QoS properties for ubicomp applications. Our current RCSM supports one QoS, situation-awareness, and we have shown how to use AOSD to expand our RCSM to support more QoS properties. We use security aspect as an example to illustrate our approach.

Currently, we are expanding the RCSM to include the security aspect and expect to generate the Security Dispatcher and some basic security components in near future.

We plan to analyze the factors related to each QoS properties that affect the performance and resource consumption, and use the analysis results to optimize resource consumption and performance. Additional QoS properties, such as real-time, will also be considered for the expansion.

Acknowledgement

This research is partially supported by National Science Foundation under grant number ANI-0123980.

References:

[1] M. Weiser, “Some Computer Science Problems in Ubiquitous Computing”, *Comm. ACM,* Vol. 36, No. 7, July 1993, pp. 75-84.

[2] S. S. Yau, Y. Wang, and F. Karim, "Development of Situation-Aware Application Software for Ubiquitous Computing Environments", *Proc. 26th IEEE Int’l Computer Software and Applications Conf. (COMPSAC 2002)*, pp. 233-238, August 26-29, 2002, Oxford, UK.

[3] I. Pyarali, D. Schmidt, and R. Cytron, “Achieving End-to-End Predictability of the TAO Real-time CORBA ORB,” *Proc. 8th IEEE Real-Time Technology and Applications Symposium (RTAS 2002)*, San Jose, CA, Sept. 2002, pp.13-22.

[4] R. Vanegas, J. Zinky, J. Loyall, D. Karr, R. Schantz and D. Bakken, “QuO's Runtime support for Quality of Service in Distributed Objects”, *Proc. IFIP Int’l Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, Sept. 1998, pp. 207-224.

[5] B. Li and K. Nahrstedt, “A Control-based Middleware Framework for Quality of Service Adaptations,” *IEEE Jour. of Selected Areas in Communications, Special Issue on Service Enabling Platform*s, vol. 17, no. 9, Sept. 1999, pp. 1632–1650.

[6] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, “A Resource Allocation Model for QoS Management,” *Proc. IEEE Real-Time Systems Symp*, Dec. 1997, pp. 298–307.

[7] S. S. Yau and F. Karim, "Reconfigurable Context-Sensitive Middleware for ADS Applications in Mobile Ad-Hoc Network Environments", *Proc. 5th Int’l Symp. on Autonomous Decentralized Systems (ISADS 2001)*, pp. 319-326, March 26-28, 2001, Dallas, USA.

[8] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing", *IEEE Pervasive Computing, Vol.* 1, No. 3, July-September 2002, pp. 33-40.

[9] J. Gray, T. Bapty, S. Neema and J. Tuck, “Handling crosscutting constraints in domain-specific modeling,” *Comm. ACM*, Vol. 44, No. 10, Oct. 2001, pp. 87 – 93.

[10] K. Lieberherr, D. Orleans and J. Ovlinger, “Aspect-oriented programming with adaptive methods,” *Comm. ACM*, Vol. 44, No. 10, Oct. 2001, pp.39-41.

[11] G. T. Sullivan, “Aspect-oriented programming using reflection and metaobject protocols,” *Comm. ACM*, Vol. 44, No.10, Oct. 2001, pp. 95-97.

[12] L. Bergmans and M. Aksit, “Composing Crosscutting Concerns Using Composition Filters,” *Comm. ACM,* Vol. 44, No. 10, Oct. 2001, pp. 51-57