Situation-Aware Contract Specification Language for Middleware for Ubiquitous Computing

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

Hoh P. In
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA
hohin@cs.tamu.edu

Abstract

Ubicomp applications are characterized as situation-aware, frequently-and-ephemerally-communicated and QoS-properties-associated. Using middleware to provide multiple QoS support for these ubicomp applications will enhance the development of the ubicomp applications. To satisfy the different QoS requirements of various applications in ubicomp environments, which are heterogeneous and resource-variant, it is important for the underlining middleware to adapt to different QoS requirements and environments. Situation-Aware Contract Specification Language (SA-CSL) specifies the QoS requirements of the applications. The specification includes requirements in situation-awareness, real-time constraints and security properties. This specification is used to customize the middleware architecture to better satisfy these requirements. SA-CSL is based on the Separation of Concern (SoC) discipline used in the Aspect-Oriented Software Development (AOSD). It specifies the crosscutting aspects of situation-awareness, real-time constraints and security property separately. Because of the object-oriented design, SA-CSL is open for incorporating new QoS properties specification.

Keywords: Ubiquitous computing, middleware, situation-awareness, Situation-Aware Contract Specification Language (SA-CSL), Reconfigurable Context-Sensitive Middleware (RCSM), Aspect-Oriented Software Development (AOSD), Quality of Service (QoS), security, real time.

1. Introduction

In ubiquitous computing (ubicomp) environments [1], where computing resources are available everywhere and a great amount of mobile devices play important roles, middleware serves as an essential infrastructure between networks and ubicomp applications. It hides the heterogeneity of the network environments and provides necessary services to ubicomp applications, such as communication, data access, resource control, and service discovery. Some ubicomp applications are situation-aware, which means that different applications use different *situation changes* to trigger different application actions. *Situation* is a set of past context attributes and/or actions of individual devices which is relevant to determine future device actions. *Context* is any instantaneous, detectable, and relevant condition of the environment or the device.

Since ubicomp environments are open and situation changes randomly, the available resources are variant and unexpected. Therefore, it is impractical to satisfy the applications' different QoS requirements (such as real-time and security) using an unchanged middleware. It is obvious that the adaptable middleware will satisfy the applications' various QoS requirements.

To facilitate the development of situation-aware ubicomp application software, we have developed the Reconfigurable Context-Sensitive Middleware (RCSM) that provides situation-analysis and response activation services [2-5]. The RCSM provides core middleware services in an Object Request Broker, and uses situation change events as triggers of inter-device communication. The situation-analyzing module of RCSM is adaptable, i.e. different application software have different situation analyzing modules generated from different situation-awareness requirements. A Situation-Aware Interface Definition Language (SA-IDL) has been developed for specifying such situation-awareness requirements [2].

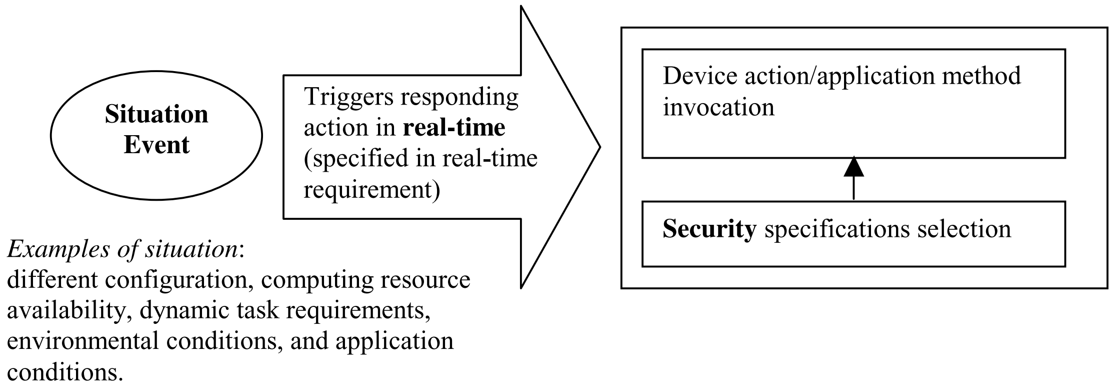
Situation Event
Triggers responding action in **real-time** (specified in real-time requirement)
Device action/application method invocation
Security specifications selection
Examples of situation: different configuration, computing resource availability, dynamic task requirements, environmental conditions, and application conditions.

Figure 1. The internal structure of SA-CSL

To expand RCSM by incorporating the QoS support for real-time and security service, we must extend SA-IDL to incorporate the specification of real-time constraints and security mechanism. This expanded language is called *Situation-Aware Contract Specification Language (SA-CSL).* The functionality of the SA-CSL is to formally specify the requirements of various applications residing on RCSM in the aspects of situation-awareness, real time and security. Then the SA-CSL specification is used in the adaptation of RCSM so that RCSM will have a better architecture to efficiently satisfy the applications' requirements.

2. Elements of SA-CSL

SA-CSL uses the presentation of situation awareness in our SA-IDL [2]. In SA-IDL, we addressed the situation-awareness by associating the situations that affect the applications and the actions the applications may take to respond to the situation. Besides specifying situation-awareness, SA-CSL is capable of specifying real-time and security requirements. For security aspect, the adoption of different security mechanisms, algorithms, techniques and policies, are associated to the device actions or application methods. For real-time aspect, all the adaptation, including the security adoption, architecture reconfiguration, and application actions, is specified with real-time requirements considering the situation-match as a situation event. Figure 1 depicts the relationship among situation awareness, real-time and security requirements.

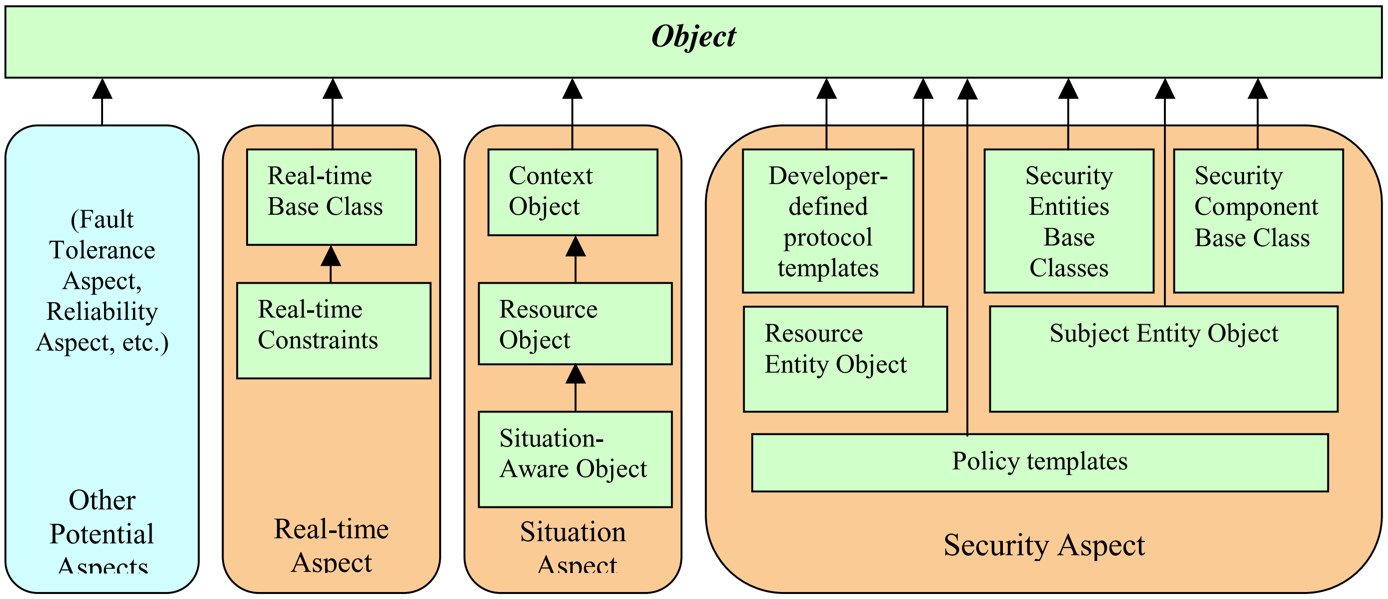

Figure 2: The object hierarchy in our SA-CSL.

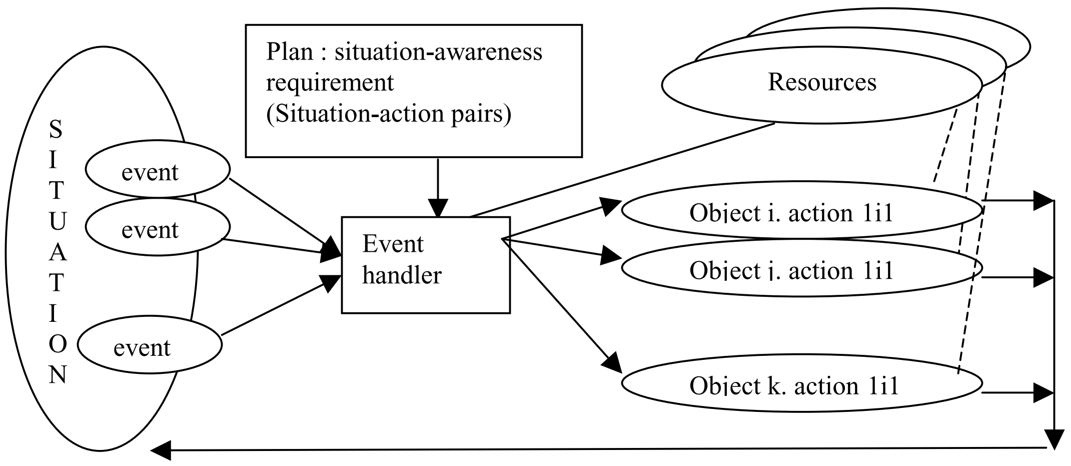

Figure 3. The conceptual model of situation-aware computing.

SA-CSL is based on the Separation of Concerns (SoC) used in Aspect-Oriented Software Development (AOSD). While the application software increases the complexity, the SoC principle is very beneficial by enabling developers to focus on one aspect at a time. Although SA-CSL includes only situation-awareness, real-time and security, it should not require much effort to expand SA-CSL to include new QoS properties, such as fault tolerance, scalability, etc. To address this, an important consideration is to support SA-CSL in an object-oriented style. As depicted in Figure 2, all the aspect specifications are presented as objects. As such, various aspects are defined independently as individual objects and new aspects can be easily added by defining new aspect object types. On the other hand, different aspects can be integrated seamlessly by defining association between aspects and application objects. Furthermore, aspect objects can be used in different applications.

3. Incorporating situation-awareness in SA-CSL

The conceptual model of situation awareness is depicted in Figure 3. The changes of situation create a set of "situation change" events. The events are captured by an event handler, which makes decision on how to react to these events. The event handler makes decisions based on an event handling plan that prepared beforehand, i.e., the specification of a set of "situation-action" pairs. Since each action requires certain resources, the event handler also checks the available resources to ensure that required resources are available before taking any action. Furthermore, the actions eventually taken are also regarded as elements that compromise new situations, and form new events. By using SA-IDL, SA-CSL defines the context attributes (and their value range), derivative, device actions, and situations. In SA-CSL, situations are used widely in describing different configurations, computing resource availability, dynamic task requirements, environmental conditions, and application conditions.

We represent *situation* as an expression on previous device-action record over a period of time and/or the variation of a set of contexts of the device over a period of time with respect to the application [2]. Situation is used to trigger further actions. This is depicted in Figure 4. To express the situations, we define the following components:

- *Context Tuple*: $\boldsymbol{<t, c_1, c_2, \ldots, c_n>}$. $\boldsymbol{t}$ is the time stamp and $\boldsymbol{c_1, c_2, \ldots, c_n}$ are a set of context attributes. Context tuple values are sampled periodically. These tuple values are the raw materials based on which we analyze situations
- *Action Tuple*: $\boldsymbol{<t, a_1, a_2, \ldots, a_n>}$, where $\boldsymbol{t}$ is the time when the action is performed and $\boldsymbol{a_1, a_2, \ldots, a_n}$ are a set of attributes about the action. They could be name, parameter types, parameter values, etc. When

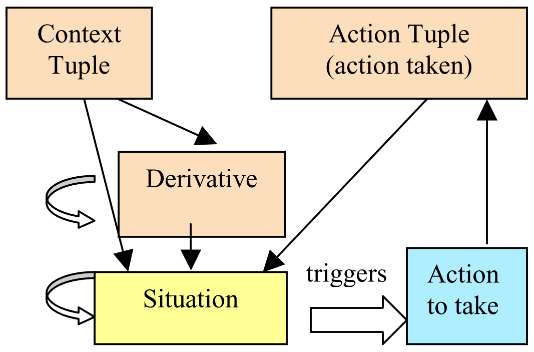

Figure 4. A situation expression system

an action is taken, the information about this action is recorded using an action tuple value.

- *Derivative*: It is the result of the analysis of contexts, such as the value of a function of several context values, whether a context value has changed, how much it has changed, etc.
- *Situation Expression*: It has the following format:

[∀,∃] t in <time range> [context, derivative, action, situation] <compare><value> (1)

We can form new situations by performing "not", "and" or "or" operations on defined situations.

To specify event-handling plan, i.e., what actions should be taken to respond to different situations, we define related rules to associate actions to situations as follows:

[activate at situation_x] action_y…. (2)

A list of such "situation_x" and "action_y" forms a plan. The event-handler monitors the situation-match events and activates the actions associated to them according to the plan.

4. Incorporating Real-Time Specification in SA-CSL

Real-time specification includes specification on events and responses. The *events* can be stimuli. These stimuli can occur repeatedly, which are normally called *event sequence*. Once an event occurs, the computation that is performed as a consequence of an event is referred as a *response*. Events can have the properties of internal, external, as well as arrival patterns, such as period. Response has properties of resources requirements, dependencies, and timing requirements, such as deadline.

The SA-CSL should be able to define the following parameters:

For event:
*event types: external (environmental) or internal or timed.
*event arrival pattern: periodic, irregular, bounded, busty arrivals, unbounded.
*mode: can be defined as a set of events.

For response:
*response: computational work that must be performed as a consequence of an event, can
consist of actions.
*actions: no resources allocation change during actions.
*ordering: order of actions that form a response.
*action attributes:
-priority/importance
-duration/deadline
-resource requirements: we defined a resource object for this parameter
-dependencies
-allocation policy: policy used to allocate resource, such as round robin...
-atomic action or not
-jitter

For Resource: To allocate resources to satisfy the real-time requirements, we need to specify the required resources of response actions. We define an object "Resource" as the base object of all specific resources objects (such as power, CPU time, and bandwidth.).

```
Resource{
  Type;
  ID;
  location;
  Required;
  Available;
  Scheduling-policy;
 }
```

Figure 5 depicts the object hierarchy of the resources.

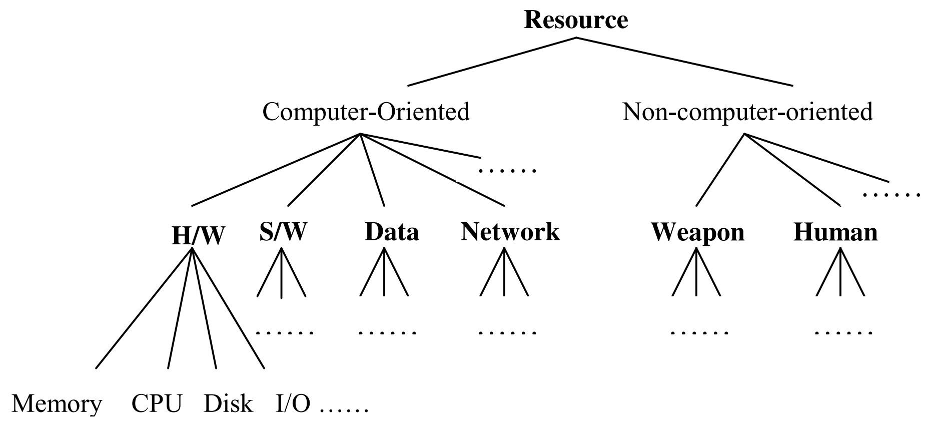

Figure 5. The resource objects hierarchy

5. Incorporating Security Specification in SA-CSL

To incorporate security requirements, the SA-CSL must be capable of defining the following elements [6,7,8]:

- *Entity:* Subjects and objects that are involved in the security management. There are several types of entities –
  - *Subject entities:* Component, application, user, device, host, group, network domain, etc. In SA-CSL, we define these entities as *subject entity objects*.
  - *Resource entities:* File, data, message, CPU, memory, etc. In SA-CSL, we define these entities as *resource entity objects*.
  - *Security entities:* We define several base classes for general security entities, such as key, certificate and role. Then, all specific security entities (e.g., X.509 certificates) extend the certificate class.
- *Action:* actions taken by entities to interact with other entities, such as sending/receiving messages, uploading/downloading files, read/write/execute files, etc. The definition of *actions* is incorporated in the definition of entities.
- *Mechanism:* the basic security operation that constitutes a specific security solution, e.g., encryption, decryption, re-keying method (for secure group communication), digest method, etc. All the security mechanisms are either provided through system library using some third party components or defined by developers. In SA-CSL, we define a *SecComp base class* to specify the common properties (e.g., type, location, vendor, size, etc.) of security components and the specific mechanism objects extend this base class to specify the interface of different security components.
- *Policy:* a high-level description of the behavior of entities. For example, a user (which is an entity) can only access a particular classified file (which is an action involving another entity) when the user is in a certain office. Policies are defined as a set of constraints, which include internal variables, the condition formed by internal variables and handlers that enforce the security requirements based on the condition.

6. An Example

In this section we will use the Smart Classroom as an example to illustrate the SA-CSL. The Smart Classroom will facilitate collaborative learning among college students in a ubicomp environment.

In a Smart Classroom, the instructor and students use their own situation-aware PDA for various learning activities, such as lecture and student presentations, discussions, and group collaborations. Students form small groups to solve a specific problem or develop a group project. During group discussions, the instructor moves from group to group to check the progress of each group or respond students' questions in their group discussions. When the instructor walks towards a group, he receives the discussion material of the group through his PDA and can be actively involved in the group's meeting immediately. When the instructor or a student is near the projector screen to show slides, the light in the classroom is automatically dimmed and the presentation material automatically distributed to the students. Students' PDAs will dynamically form mobile ad hoc networks for group meetings. Each PDA monitors its situation (locations of PDAs, noise, light, and mobility) and uses situation change event to trigger communications among the students and the instructor

```
SecurityConstraint {  //a security requirement object
            Action in;
            Action out;
            Mechanism m;
            assert in.result == out.input;
            [on satisfying] out.input=m(in.result);
} Sec1;

QoS-RealTime {  // a real-time constraint object
            Int duration;
            Int importance;
}RTC1;

Resources{  // a resource object
            Component (bandwidth, memory, CPU);
}resource1;

//instantiate the real-time, security requirement, and
//resource objects needed by download action
rtDownload = new RTC1 (n, 1}
secureDownload = new Sec1 (download,
student.sendDiscussion, PublicEncryption);
resourceDownload = new Resources(new
bandwidth(24), new memory(300), new CPU(64));

Situation-aware-object {
      Situation situ1;
      //situ1 represents that the instructor is moving to
      //group k. Refer to [2] for detailed specification;
      [incoming] [activate at situ1] download ()
      RequiredResources resourceDownload;
      WithSecurityConstraint secureDownload;
      WithRealtimeConstraint resourceDownload;
}instructor;
```

Figure 6. SA-CSL sample code

for group discussion, and automatic distribution of presentation materials, if any.

Let us consider the following scenario: During group discussion, when the instructor is moving to a specific group, his PDA will automatically download this group's discussion material. The download should be started within *n* seconds after the situation is true, and the discussion material must be secured using secure mechanism *m*. Figure 6 shows the sample code of SA-CSL for this scenario.

7. Comparison with Current Technology

Our SA-CSL is to specify the QoS requirements of the ubicomp applications. The requirements include aspects of situation-awareness, real time and security. The middleware for ubicomp applications uses the SA-CSL specification in adaptation of the architecture to satisfy the applications requirements. To compare SA-CSL with existing techniques, we observe the following:

- On situation-awareness: In situation-aware computing, SA-CSL is the only language that enables developers to specify the situations in interface level and then automatically generate the situation analysis module [2]. Currently, other methods dealing with situation-awareness either directly develop the situation analyze module manually [9,10] or manually build a set of context processing component for later integration of synthesis [11].

- On general QoS specification: QoS Description Languages (QDL) [12] also addresses the middleware QoS: dependability, real-time and security, but requires the use of a separate language for each aspect. QoS Modeling Language (QML) [13] is a general specification language that is not limited to a specific QoS aspect or application domain. Neither QDL nor QML support situation-awareness. Our SA-CSL can be used to specify real-time and security aspects integrated with the desired situation-awareness.

- On security specification: Although WS-Security [14], Ponder Policy Specification language [6] and Secure Operations Language (SOL) [15] have good security specification, they do not address the relation of situation-awareness and security. Gaia [16] provides situation-awareness access control by defining a space role for an active space and associating an access list with each service in the space. Since our specification language associates security specification with situations and actions, it is more powerful to express situation-aware security. Furthermore, since our specification language specifies the security mechanisms that are going to be used, it is possible to integrate third-party security components in RCSM. Moreover, it facilitates the trade-off analysis for different QoS aspects (such as real-time and security) because what security mechanisms are used is clearly defined.

8. Conclusion

The SA-CSL is used to facilitate the middleware adaptation by specifying various aspects of requirements of the applications software that resides on the middleware. SA-CSL separates the specification of various crosscutting aspects: situation awareness, real-time constraints, and security requirements. Each aspect is represented as a multiple-attribute type. SA-CSL enables the application developers to specify the QoS requirements at design phase and deliver the QoS-related service to the middleware. We have discussed the design consideration of the SA-CSL and we are in process of constructing production rules and compiler of the SA-CSL, and developing the approach to adapting middleware architecture according to SA-CSL specification.

A potential drawback of our approach is that the SA-CSL defines the QoS constraints in a single level – operation level, that is all the QoS constraints are associated with operations of an application objects. This may not cover all the QoS specifications, such as the QoS constraint for a whole interface or for a stream. Second, the QoS property is defined as a multi-attribute type. The issue on whether this form of QoS definition is sufficient for all the QoS properties should be further explored.

Acknowledgment

This research was supported in part by the National Science Foundation under grant number ANI-0123980.

References:

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

[2] S. Yau, Y. Wang, and F. Karim, "Developing Situation-Awareness in Middleware for Ubicomp Environments," *Proc. 26th Int'l Computer Software and Applications Conf. (COMPSAC 2002),* pp. 233-238.

[3] S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing,* 1(3), July-September 2002, pp. 33-40.

[4] S. S. Yau and F. Karim, "Adaptive Middleware for Ubiquitous Computing Environments", *Design and Analysis of Distributed Embedded Systems, Proc. IFIP*

17th World Computer Congress, August 2002, Vol. 219, pp. 131-140.

[5] S. S. Yau and F. Karim, "Context-Sensitive Middleware for Real-time Software in Ubiquitous Computing Environments", *Proc. 4th IEEE Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC 2001)*, May 2001, pp. 163-170.

[6] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language", *Proc. Int'l Workshop on Policies for Distributed Syst. and Networks (POLICY), Springer-Verlag LNCS 1995*, Bristol, UK, Jan. 2001, pp. 18-39.

[7] A. Heydon, M. W. Maimone, J. D. Tygar, J. M. Wing, and A. M. Zaremski, "Miro: Visual Specification of Security", *IEEE Trans. on Software Engineering*, Vol. 16. No. 10. October 1990, pp. 1185-1197

[8] L. C. Aiello, and F. Massacci; "An executable specification language for planning attacks to security protocols", *IEEE Computer Security Foundation Workshop*, P. Syverson, (ed.) IEEE Computer Society Press, 2000, pp. 88-103.

[9] A. Chen et al, "A Support Infrastructure for Smart Kindergarten", *IEEE Pervasive Computing*, 1(2), June 2002, pp. 49–57.

[10] G.D. Abowd, "Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment," *IBM Systems. J.*, vol. 38, no. 4, Oct. 1999, pp. 508–530.

[11] A.K. Dey and G. Abowd, "The Context-Toolkit: Aiding the Development of Context-Aware Applications," *Proc. Conf. on Human Factors in Computing Systems (CHI), USA, 1999*, pp 434–441.

[12] P. Pal, et al., "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration," *Proc. 3nd IEEE Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC 2000)*, March 15-17, 2000, Newport Beach, CA. pp. 310-319.

[13] S. Frolund and J. Koistinen, "Quality of Service Specification in Distributed Object Systems Design," *Proc. 4th USENIX Conf. on Object-Oriented Technologies and Systems (COOTS)* Santa Fe, New Mexico, April 27-30, 1998. http://www.usenix.org/publications/library/proceedings/coots98/full_papers/frolund/frolund.pdf

[14] Web Service Security (WS-Security) version 1.0, April 5, 2002 http://www.verisign.com/wss/wss.pdf

[15] R. Bharadwaj, "SOL: A Verifiable Synchronous Language for Reactive Systems", *Proc. Synchronous Languages, Applications, and Programming (SLAP'02), ETAPS'2002*, April 13, 2002, Grenoble, France. http://www.inrialpes.fr/bip/people/girault/Slap02/programme.html

[16] G. Sampemane, P. Naldurg, and R. H. Campbell, "Access control for Active Spaces", *Proc. Annual Computer Security Applications Conf. (ACSAC2002)*, Las Vegas, Nevada, Dec 9-13 2002. http://choices.cs.uiuc.edu/gaia/papers/acsac02-space-sec.pdf