Development and Runtime Support for Situation-Aware Application Software in Ubiquitous Computing Environments

Stephen S. Yau, Dazhi Huang, Haishan Gong, Siddharth Seth
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-8809, USA
{yau, dazhi.huang, haishan.gong, sidseth}@asu.edu

Abstract

Due to the dynamic and ephemeral nature of ubiquitous computing (ubicomp) environments, it is necessary that application software in ubicomp environments is situation-aware (SA) and should be adaptable to both users' situation changes and the requirement changes. Reconfigurable Context-Sensitive Middleware (RCSM) has been developed to provide development and runtime support for SA software in ubicomp environments, but do not provide runtime support for on-demand context acquisition, action scheduling and dynamic reconfiguration of SA software, and development support that maximize the reusability. In this paper, the development and runtime support provided by RCSM have been substantially expanded to greatly simplify the development of situation-aware application software, and to achieve reusability and runtime reconfigurability simultaneously.

Keywords: Situation-aware application software, ubiquitous computing, Situation-Aware Interface Definition Language (SA-IDL), RCSM, development support, runtime support.

1. Introduction

Ubiquitous computing (ubicomp) environments provide access to information and computing resources for users at any time and anywhere. An application software running on the mobile and wearable devices in ubicomp environments often needs to be situation-aware (SA) in order to relieve users from adjusting the devices operations due the dynamic changes of the situations of devices. We consider a *situation* as a set of past contexts and/or actions of individual devices relevant to future device actions, and a *context* is any instantaneous, detectable, and relevant condition of the environment or the device [1]. In general, SA application software in ubicomp environments operates with the following three phases:

- ***Context acquisition*** from ambient environments.
- ***Situation analysis*** to determine the situations.
- ***Proper action triggering*** based on the current situation. When multiple actions are triggered simultaneously, scheduling of triggered-actions is needed to generate the proper triggering sequence of these actions to ensure the correctness of the SA application software and satisfy the real-time requirements of these actions.

From these three phases, SA application software needs to have the capabilities of context acquisition, situation analysis, wireless ad hoc communications and action scheduling. Developing SA application software with these capabilities is challenging due to the severe resource constraints of ubicomp devices and the ephemeral and heterogeneous ubicomp environments. Hence, development and runtime support is needed to simplify the development and improve the performance of SA application software with these capabilities. Moreover, it is desirable that the development support can maximize the reusability of SA application software, and the runtime support can facilitate the runtime reconfiguration of SA application software to adapt to the changing ubicomp environments.

Reconfigurable Context-Sensitive Middleware (RCSM) [1-3], which was developed, cannot provide development support for maximizing the reusability and runtime support for on-demand context acquisition, action scheduling and runtime reconfiguration of SA application software. In this paper, we will present greatly expanded development and runtime support in RCSM. Our expanded development support includes situation-

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

awareness requirement specification with more reusable context representation and new notations for action scheduling requirements, and code generation. Our expanded runtime support will include on-demand context acquisition, action scheduling, SA requirement management for reusability, and runtime reconfiguration of context acquisition and situation analysis. They will greatly simplify the development of situation-aware application software while achieving reusability and runtime reconfigurability simultaneously.

2. Desirable Development and Runtime Support

Based on the discussion in Sec. 1, the following development and runtime supports for SA application software in ubicomp environments are desirable:

D1) Context Acquisition Transparency: Runtime support should provide well-defined interfaces for application developers to retrieve contexts from various sources without knowing how contexts are collected.

D2) On-demand Context Acquisition: Development support should allow application developers to specify their requirements on contexts. Runtime support should provide the capabilities to discover and collect contexts based on the needs of SA application software.

D3) Situation Analysis Transparency: Runtime support should provide the capability to analyze collected context data to determine the situation, and notify SA application software when the situation changes. Development support should allow application developers to specify the situations of their interests.

D4) Reusability: Runtime support should provide the capability to share the situation and context specifications among SA application software to improve reusability.

D5) Communication Transparency: Runtime support should provide transparent communication support for SA application software to communicate with other SA application software over infrastructure or ad hoc wireless and wired networks.

D6) Scheduling: Development support should allow application developers to specify scheduling requirements on situation-triggered actions. Runtime support should provide the capability to schedule situation-triggered actions based on their scheduling requirements.

D7) Reconfigurability: Runtime support should allow users or application developers to change the situation-awareness requirements of SA application software.

D8) Efficiency: Runtime support should conserve resources, such as energy, memory, CPU time and network communication bandwidth.

3. Current State of the Art

In the last several years, substantial research [4-10] has been done in context-aware computing. Several frameworks, toolkits and infrastructures have been developed for providing support to context-aware application development. Notable results include ParcTabs, Stick-e Notes, CALAIS, CoolTown, Context Toolkit, Context Fabric and MobiPADS.

ParcTabs [4] and CALAIS [5] support acquisition of contexts (**D1**) about users and devices. Context Toolkit [6] aims at providing architectural support for context-aware applications (**D1** and **D4**). CoolTown [7] aims at supporting applications that display contexts and services to end-users (**D1**). Context Fabric [8] aims at making context-aware applications easy to evolve and maintain by providing a service-based infrastructure (**D1** and **D7**), and uses a context specification language (CSL) for declaring and processing context needs [9] (**D2**), i.e., writing context queries from the device's local context service. Stick-e Nodes [10] also provides mechanisms for indicating what contexts are needed and specifying actions to be taken based on a particular combination of contexts (**D1 and D2**). MobiPADS is a reflective middleware that is designed to support dynamic adaptation of context-aware services based on which application's runtime reconfiguration is achieved (**D1**, **D2**, **D4** and **D7**). None of them can satisfy **D3**, **D5** and **D6**.

Our previous RCSM [1-3] provides the following support for SA application software:

1) Development support. A declarative Situation-Aware Interface Definition Language has been developed in RCSM to *specify SA requirements*. The SA-IDL compiler *generates* an application specific Situation-Aware Adaptive Object Containers (SA-ADC) based on the SA-IDL specification of SA application software.

2) Runtime support. RCSM Object Request Broker (R-ORB) provides efficient and transparent supports for *context acquisition* and *wireless ad hoc communication*. SA-ADCs can *analyze* context and action history to recognize situations, and *trigger* proper actions based on situations.

These development and runtime supports by our previous RCSM provide efficient (**D8**) and transparent context acquisition (**D1**), situation analysis (**D3**), and wireless ad hoc communications (**D5**) to SA application software. However, they could not satisfy requirements **D2**, **D4**, **D6** and **D7**.

4. Expanded Development and Runtime Support in RCSM

To overcome the limitations of our previous RCSM, we have greatly expanded the development and runtime support in RCSM with all the desirable features discussed in Sec. 2.

```
/* Specify context class hierarchy*/
RCSMContext class Base {
    // define attributes here, if any
}
RCSMContext class Location extends Base {
    string location_name;
}
RCSMContext class GPS_Location extends Location {
    double latitude;
    double longitude;
}
RCSMContext class Mailing_Location extends Location {
    string street;
    string city;
    string zipcode;
}
```

Figure 1. Representing location contexts using our improved SA-IDL.

4.1 Expanded Development Support

Our expanded development support includes an improved SA-IDL and its compiler for SA requirement specification and automated code generation.

To develop SA application software, application developers need to first identify the SA requirements, which include what contexts need to be collected, what are the situations of interest, what actions should be triggered in certain situations and how the triggered actions should be scheduled. Then, application developers can use SA-IDL to specify the SA requirements and use the SA-IDL compiler to generate the application skeleton for the SA application software.

❑ **SA requirement specification using the improved SA-IDL.**

SA requirement specification includes context representation, situation representation and situation rule representation. The improved SA-IDL has the following two major features:

(1) Using object-oriented context representation (**D4**), which is a concise and reusable way to represent contexts, and can facilitate application evolvement and reconfiguration.

In ubicomp environments, sources of context are heterogeneous, i.e., a context acquired from different sources may be in different formats. For example, location information could be described in latitude-longitude format when acquired from GPS, or in street_number-city-zip_code format when acquired from a location server.

As mentioned in Sec. 2, the context representation in our previous SA-IDL is not concise and reusable. Furthermore, the context representation in previous SA-IDL cannot show the relationship among different contexts, which makes context discovery difficult. We have adopted object-oriented modeling technique to represent contexts in the improved SA-IDL. Figure 1 shows a set of context classes with inheritance relationships. Using object-oriented modeling technique will result in better design due to added flexibility, extensibility and ability to evolve, in both incorporating modifications in existing contexts and adding new contexts.

(2) Incorporating action scheduling requirement specification in situation rule representation (**D6**).

Situation rule representation indicates the situation and actions mapping relationship. When a situation is recognized, one or more actions may be triggered. The improved SA-IDL provides necessary abstractions for application developers to specify action-scheduling requirements. The scheduling requirement of an action has two properties: "within" indicates the deadline that the action must be triggered. The number associated with the "priority" of the action represents the action's priority, the higher value the higher priority. The new runtime support for action scheduling, which will be explained later, is base on these two features of SA-IDL.

Figure 2 shows an example for SA-IDL specification of an SA application.

```
RCSMContext Class Time extends Base
  {  int weekday;  int currentTime; }
RCSMContext Class Location extends Base
  { string room; }
RCSM Context Class ClassSchedule extends Base
  { string className;  int weekday;  int start_time;}
    … …
RCSM Context Acquisition { Time {frequency = 4;}}

RCSMSARule BeforeLecture {
   PrimitiveSituation approachingClassTime
   ([-1,0] ClassSchedule.start_time - Time.currentTime < 3600);
   PrimitiveSituation inOffice
   ([-1,0] Location.room == MyInfo.myOffice);
   PrimitiveSituation myDesktopIsAround
   ([-1,0] NeighborDevice.ID = = MyInfo.myPC);
   … …
   CompositeSituation preparation
   (approachingClassTime && inOffice && myDesktopIsAround
   && … … );
   ActivateAt preparation {
   [incoming] void download_slides(string) [within 3][priority 1]
   }
}
```

Figure 2. An SA application specification using our improved SA-IDL

❑ **Automated code generation using the compiler for the improved SA-IDL.**

In order to greatly improve the reusability and reconfigurability of SA application software and reduce

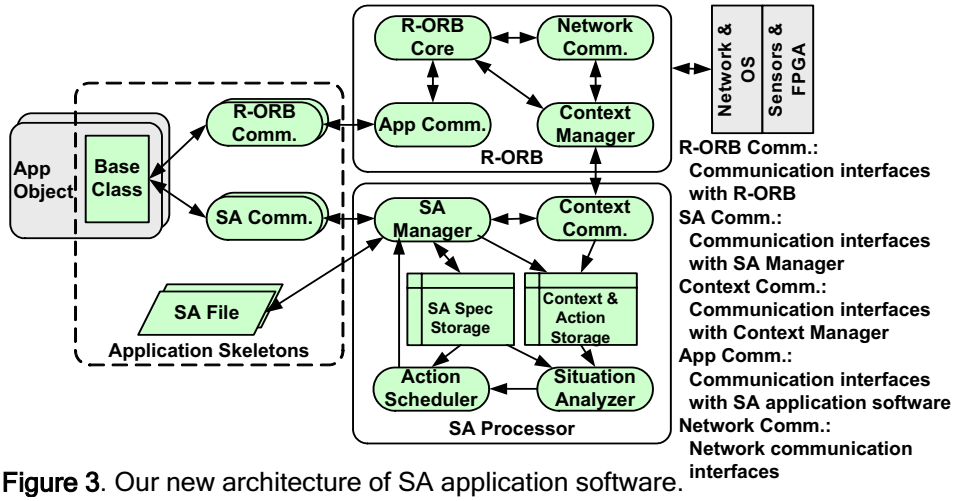

Figure 3. Our new architecture of SA application software.

unnecessary memory and communication overhead, we have developed a new architecture of SA application software, which completely decouples application functional components and the situation-processing components. Figure 3 depicts this architecture. Decoupling application functional components and the situation-processing components will greatly enhance code reusability since different SA application software can share the same situation-processing components. Such decoupling will also greatly enhance the reconfigurability of SA application software because any changes on an application's functional components will not affect its situation-processing components, and vice versa.

Application skeletons in Figure 3 are generated the improved SA-IDL compiler by compiling the SA-IDL specifications. They simplify the development of SA application software by providing extensible object skeletons for application developers to implement the applications' functionalities. An Application Skeleton consists of a base class, R-ORB Comm., SA Comm., and a SA file:

- The base class defines the interface of an SA application object, how an action (method) of the SA application object is triggered by SA Processor when certain situation is recognized, and when the SA application object needs to communicate with a remote object using R-ORB. Once application skeletons are generated, application developers only need to extend the base classes to implement the functionalities for SA application objects.
- The R-ORB Comm. and SA Comm. provide standard interfaces with R-ORB and SA Processor to support the interactions between the SA application object and SA Processor or R-ORB, and mask the complexity of these interactions from application developers.
- The SA file is a formatted XML file that contains the SA requirements of the SA application object. SA file is used by SA Processor in runtime.

4.2 Expanded Runtime Support

Our expanded runtime support includes support for runtime reconfigurable and on-demand context acquisition (**D2** and **D7**), situation analysis based on runtime reconfigurable SA requirements (**D7**), SA requirement management for reusability (**D4**) and action scheduling (**D6**). The expanded runtime support is provided by SA Processor and R-ORB in the expanded RCSM.

❑ **Using SA Processor and R-ORB to facilitate the execution and reconfiguration of SA application software.** In runtime, SA Processor and R-ORB are running as service processes. R-ORB provides interfaces for context discovery, collection and propagation. SA Processor manages SA requirement specifications, maintains the context and action history, recognizes situations based on the context and action history, and schedules actions to be triggered. Specifically, R-ORB and SA Processor facilitate the execution of SA application software as follows:

1) When the SA application starts running, it sends a registration message to SA Processor. The registration message contains the application object's name, communication port of its *SA Comm.*, and the location of the application's *SA file*.
2) Once SA Processor receives a registration message, it will read the new SA requirement specifications from the application's SA file, and use its SA requirement management capability to check for new contexts, situations and situation rules.
3) If some contexts and situations are being used by other SA applications currently running on the same device, SA Processor will reuse (**D4**) these contexts and situations for the new SA application.
4) If new situations and situation rules are found, SA Processor will add these new situations and situation rules into its SA Spec Storage, and analyze context and action history for recognizing the new situations.
5) If new contexts requirements are found, SA Processor will send them to R-ORB. R-ORB will reconfigure (**D7**) its context acquisition capability based on the new requirements (**D2**) from SA Processor, and send the updates of required contexts to SA Processor.
6) When SA Processor receives updated contexts from R-ORB, it will update and analyze the context and action history.

7) If a situation is recognized, SA Processor will trigger the proper actions based on the corresponding situation rules. If multiple actions can be triggered simultaneously, SA Processor will schedule these actions based on their scheduling requirements in the situation rules.

The reconfiguration of SA application software is supported by SA Processor. The SA requirement management capability provided by SA Processor allows application developers or users to query and update SA requirements in runtime. Once SA Processor receives an updated SA requirement, it will start a similar process as 2) – 7) above to handle the updated requirement.

❑ **Situation analysis, action scheduling and SA requirement management using SA Processor.** Our previous runtime support for situation analysis is provided by application specific SA-ADCs [1, 2]. Based on the discussion in Sec. 2, our previous SA-ADCs cannot satisfy **D4**, **D6** and **D7**, and cause unnecessary memory and communication overhead. To overcome these limitations, we have redeveloped the application-independent SA Processor to address reliability (**D4**), scheduling (**D6**), reconfigurability (**D7**), as well as provide runtime situation analysis support for multiple SA application software simultaneously. SA Processor consists of the following components:

- ***SA Spec Storage*** stores all the SA requirement specifications of SA application software running on the device. It allows SA specifications to be shared and reused (D4) in different SA application software running on the device.
- ***Context and Action Storage*** stores the collected context data and action history of SA application software running on the device.
- ***Context Comm.*** notifies *Context Manager* in R-ORB of contexts requested by SA application software, retrieves updated context data from R-ORB and updates *Context and Action Storage*.
- ***SA Manager*** provides support for SA requirement management. It accepts registrations from SA application software, reads the applications' *SA Files*, updates *SA Spec Storage* by adding new SA requirement specifications from the applications' *SA File*, sends context requirements to *Context Comm.*, and notifies SA application software of the situation and the actions to be taken. SA Manager also provides query/update interfaces to support changing SA requirement specifications in runtime (**D7**).
- ***Situation Analyzer*** provides support for reconfigurable (**D7**) situation analysis. It checks *Context and Action Storage* periodically for updated context and action history, recognize the situation based on SA requirement specifications in SA Spec Storage, and sends the recognized situation to *Action Scheduler*. In our previous SA-ADCs, the logic for recognizing situations is hard-coded and generated by our previous SA-IDL compiler, and hence the situation analysis capability provided by SA-ADCs cannot be reconfigured in runtime. To address the reconfigurability, the *Situation Analyzer* in our new SA Processor adopts another approach: it reads the SA requirement specifications in *SA Spec Storage* in runtime, and uses a first-order logic rule processing engine to recognize situations. Therefore, when an SA requirement specification changes or new SA requirement specifications are added into *SA Spec Storage*, *Situation Analyzer* will automatically use the new requirements for future situation analysis.
- ***Action Scheduler*** provides support for action scheduling (**D6**). Upon receiving a recognized situation from *Situation Analyzer*, *Action Scheduler* lookups *SA Spec Storage* to identify actions to be triggered under the recognized situation and these actions' scheduling requirements, schedules these actions based on their priorities and deadlines, and sends the results to *SA Manager*. The results will include the recognized situation, actions to be triggered and the schedule of triggered actions. Similar to *Situation Analyzer*, the action scheduling can be reconfigured (**D7**) in runtime by changing the actions' scheduling requirements in SA rules.

❑ **Context acquisition using R-ORB.** Our previous R-ORB consists of Context Manager, R-ORB Core, App Comm. and Network Comm. Context Manager collects context data from various sensing units, including local or remote sensors for monitoring environments, and software routines for monitoring system status and user contexts. R-ORB Core, App Comm. and Network Comm. provide wireless ad hoc communication support for SA application software. Based on the discussion in Sec. 2, we can see that the previous R-ORB cannot satisfy **D2** and **D7**.

To satisfy **D2** and **D7**, an adaptive, lightweight and energy-efficient context discovery protocol, R-CDP [11], has been developed and used in our new Context Manager, which manages the registrations of local sensing units, discovers remote sensing units, retrieves context data based on requirements from *Context Comm.* in SA Processor, and sends collected data to *Context Comm.* as follows:

1) When a new local sensing unit starts running, it will register in *Context Manager* by sending the information on its sensing capability, including the contexts it has collected, the methods of acquiring contexts, and the range of supported frequency of context acquisition. But, a sensing unit will not be activated until a SA application requests the contexts provided by the sensing unit.

2) When a SA application starts running, its requirements on contexts will be extracted and sent to *Context Manager* by SA Processor.
3) *Context Manager* will check whether any local sensing units can provide the required contexts, and initiate the context discovery process to find remote sensing units that can provide the required contexts, if necessary.
4) If all required contexts are found, *Context Manager* will send a "registration success" message and updates of the required contexts to *Context Comm.* in SA Processor. Otherwise, it will send *Context Comm.* a "registration fail" message to terminate the execution of the SA application software.

A set of sensing units, which were developed for our previous R-ORB, have been modified to support the above context discovery process, and used as default sensing units for our expanded R-ORB. The set of sensing units includes hardware/software sensors for collecting environmental contexts (time, location, light, noise and motion), and software routines for collecting system and network status (memory, battery power, network bandwidth, etc.). Sensing units for collecting user contexts are usually domain- or application- specific, and hence they need to be provided by domain experts or application developers.

4.3 Implementation of the Expanded Development and Runtime Support of RCSM

We have implemented the improved SA-IDL compiler in C, which can generate application skeletons implemented in eMbedded VC++ executed on Windows CE platforms. The SA Processor and R-ORB in the expanded RCSM are also implemented in eMbedded VC++. Currently, we are developing demonstration applications using our expanded development and runtime support, and evaluating the performance of SA Processor and R-ORB in the expanded RCSM.

5. Conclusions and Future Work

In this paper, we have presented our expanded development and runtime support provided by our new RCSM for situation-aware application software in ubicomp environments. We have identified a set of desirable development and runtime support, and discussed how our new RCSM provides such desirable support, which includes situation-awareness requirement specification with more reusable context representation, new notations for context acquisition rate and action scheduling requirements, code generation, runtime reconfigurable and on-demand context acquisition, situation analysis based on runtime reconfigurable SA requirements, SA requirement management for reusability and action scheduling. The above support can greatly simplify the development and facilitate the execution and reconfiguration of SA application software. Future work includes improving the expressive power of SA-IDL, developing a model-based approach for verification of SA-IDL specification, fuzzy situation rule processing, and development and runtime support for security and privacy.

Acknowledgment

This work was supported in part by National Science Foundation under grant number ANI 0123980.

References

[1] 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)*, August, 2002, pp. 233-238.

[2] S. 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.

[3] S. S. Yau and F. Karim, "An Energy-efficient Object Discovery Protocol for Context-Sensitive Middleware for Ubiquitous Computing", *IEEE Trans. on Parallel and Distributed Systems*, vol. 14(11), November, 2003, pp. 1074-1084.

[4] B. Schilit, "System architecture for context mobile computing," Unpublished doctoral dissertation, Columbia University, 1995.

[5] B. J. Nelson, "Context-Aware and Location Systems", PhD thesis, University of Cambridge, Jan. 1998 http://www.sigmobile.org/phd/1998/theses/nelson.pdf

[6] A. K. Dey and G. D. Abowd "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human-Computer Interaction*, vol. 16(2-4), 2001, pp. 97-166.

[7] D. Caswell and P. Debaty, "Creating Web representations for places," *Proc. 2nd Int'l Symp. on Handheld and Ubiquitous Computing (HUC2K)*, 2000, pp. 114-126.

[8] J. Hong and J.A. Landay "An Infrastructure Approach to Context-Aware Computing," *Human-Computer Interaction*, vol. 16(2-4), 2001, pp. 287-303.

[9] J. Hong, "The Context Fabric: An Infrastructure for Context-Aware Computing," *Proc. ACM CHI '02 extended abstracts on Human Factors in Computing Systems*, vol. 2, 2002, pp.554-555.

[10] P. J. Brown, "The stick-e Document: a framework for creating context-aware applications" *Proc. Electronic Publishing 1996*, pp.259-272.

[11] S. S. Yau, D. Chandrasekar and D. Huang, "An adaptive, Lightweight and Energy-Efficient Context Discovery Protocol for Ubiquitous Computing Environments," *Proc. 10th Int'l Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)*, May 26-28, 2004, pp. 261-267.

[12] S. S. Yau, S. Gupta, F. Karim, S. Ahamed, Y. Wang and B. Wang, "A Smart Classroom for Enhancing Collaborative Learning Using Pervasive Computing Technology", *Proc. 6th WFEO World Congress on Eng. Education & 2nd ASEE Int'l Colloquium on Eng. Education (ASEE2003)*, June 2003.