A Group-Aware Middleware for Ubiquitous Computing Environments

Dongman Lee, Seunghyun Han, Insuk Park, SaeHoon Kang, Kyungmin Lee, Soon J. Hyun, Young-Hee Lee, and Geehyuk Lee
Information & Communications University,
Yusung-Ku, Taejon, 305-714, Korea
{dlee, dennis, ispark, kang, kmlee, shyun, yhlee, geehyuk}@icu.ac.kr

Abstract

In this paper, we present, Active Surroundings, a group-aware middleware infrastructure for ubiquitous computing environments. Our system focuses on two key issues: group-awareness and transparent application reconfiguration. To achieve these goals, Active Surroundings is composed of four key component: environment sensing for abstraction of environment status and changes and tracing movements of a group of users, context management for collection, identification, and representation of user intention and resolution of group context conflict, context-aware service discovery for finding appropriate service based on context information, and dynamic reconfiguration support for dynamic reconfiguration of service objects with minimal overhead.

Key words: Ubiquitous Computing, Group-Awareness, Context-Awareness, Service Transparency, Middleware.

1. Introduction

With the advancement of the network and computer technology, they are to become commodity in our daily lives such as electricity and telephone. However, novice users still face with difficulty since they are asked to reconfigure their task when the existing environment changes or a new environment is introduced due to user's location change. A main goal of ubiquitous computing is to make the system, on behalf of a user, dynamically adapt to the changes of a user environment, allowing the user just focusing on his activity [6]. For this, applications in a ubiquitous environment are required to dynamically and transparently adapt to the changes of a user context and/or an environment due to user mobility and addition or deletion of devices and services, for instance.

Several software infrastructures [1, 2, 4, 6] have been proposed for development of ubiquitous computing applications. Context awareness and dynamic reconfiguration are key considerations in ubiquitous computing middleware design among others such as service discovery, robustness, privacy, etc [2]. Context awareness is the ability of understanding user's intention, responding to the changes in the environment. Dynamic reconfiguration of an application is required to make applications transparently adapt to context changes. In real world, users usually interact with others when they perform their task. During the interaction, their intentions may conflict and it is difficult to pre-define all the conflict situations since interaction among people is dynamic. And an application has to be designed to reconfigure itself for adapting to a new context dynamically. However, the existing systems either provide no support of the group context or require applications to specify the group context explicitly. In these systems, binding the required services in the new environment to a user application has to be done either in a pre-defined way or diminished to only available services when appropriate services are not available in the environment.

We propose Active Surroundings as a ubiquitous computing middleware infrastructure. The proposed middleware supports not only key required services of ubiquitous computing but also allows applications to transparently deal with dynamics of group context and seamlessly adapt to context changes. We assume that the environment is inherently composed of heterogeneous devices. The proposed system aims to support a group-aware ubiquitous computing environment where the system minimizes user involvements when users move from one environment to another. Active Surroundings is composed of four components; environment sensing, context management, dynamic reconfiguration support and context-aware service discovery, to enable users to be free from intrusion by system or devices in shared ubiquitous computing environments. The environment-sensing component aims to sense context changes in an active surrounding. The context management component focuses on describing context information and deciding context changes in an active surrounding. The context-aware service discovery component dynamically discovers and selects appropriate services according to the context information such as user's profile, location and computing environment nearby. The dynamic reconfiguration support component allows applications to be polymorphically transformed based on the current available resources and services without requiring user intervention.

2. Related Works

Several prototypical systems for ubiquitous computing environments are proposed. Gaia [15] is intended for a middleware infrastructure for people-centric ubiquitous computing environments. It focuses on resource-awareness and user-centric environments. In terms of group-contexts, The Gaia can handle conflicts occurred by multiple actions triggered in a certain contexts. It uses a priority to resolve a conflict, that is, if there were two conflicting actions, the actions with the higher priority wins. How-ever, only the statically pre-described context at the time of designing the applications can be handled. Gaia exploits the MVC model to separate application logic and its presentation. It enables applications to be dynamically bound with various output devices depending on the user's context. The binding information is represented with application generic description (AGD) and application customized description (ACD). Gaia provides both user-initiated and automatic mappings for binding. However, it requires applications to be totally rebuilt-up with pre-programmed coordinator when context change due to, for instance, user mobility, occur.

Aura [5] aims to provide user distraction-free computing environment where people can get services or perform their jobs without interventions by system or environments. It views user attention as one of scare resources in ubiquitous computing environment. It challenges to minimize distraction to users. It focuses on architectural issues for service composition and adaptation than context-awareness. The functionality of the context observer merely depends on the sensors deployed in the environment. Aura does not appear to support the group context. It uses task-driven approach to recon-figure applications to support nomadic users. When environments changes occur, tasks are migrated to current environment. However, it does not specify how to sup-port task composition at run-time, how users and/or application developers modify a task, and how to support non-intrusiveness.

Reconfigurable Context Sensitive Middleware (RCSM) [19] is a middleware designed to facilitate applications that require context-awareness and ad-hoc communication. It provides an object-based framework for supporting context-sensitive application. It has two key features. First, it is based on CORBA but core parts are implemented in FPGA in order to achieve high performance. Second, it is initiated based on specific environmental conditions, rather than explicit application invocation. However, it does not focus on activities of group of users in ubiquitous computing environments. It uses SA-IDL to define situations and SA-ADC is automatically generated with the SA-IDL compiler. However, the developer is responsible to generate the code of the situation-aware object, which assumes that the developer knows possible situations in heterogeneous environments.

The CARISMA system also proposed a runtime conflict-resolving mechanism [1]. It employs a particular type of sealed-bid auction. However, the resolution for its inter- and intra-conflict is just a selection of pre-described candidate services, that is, it can only suggest a static adaptation policy to the applications but no adaptation policy is generated dynamically. Solar proposed the operator graph which creates new contexts by aggregating, merging, transforming and filtering given contexts [3]. It does not support specification of user applications. For user applications, it can incorporate Java programs. Korpipaa et al. and Mantyjavi reported context description gathered from sensors attached to a mobile device using ontology [21]. The context information is used to derive higher level contexts that are more conceptual and insensible using a naïve Bayes classifier. It also provided application programming interface (API) for the context management to help develop user applications. User applications are con-trolled by a Fuzzy rule base and adapt to given contexts according to the result of AND operation of fuzzy membership of contexts. Many research projects on the con-text management describe their environments using ontologies proposed by the Se-mantic Web research groups such as DAML+OIL and OWL [1], [12], [14], [15]. Ontology is a kind of classification for entities or contexts which consist of an environment, a knowledge base for inferring the intention of users and higher level con-texts, a common terminology and a shared set of concepts of various contexts to sup-port interoperability between context-aware applications.

CAP focuses on coordination of the adaptive behavior of multiple applications in order to achieve a goal on a system-wide level [4]. It uses event-driven policy description language derived from event calculus logic to specify policies. However, it does not support for distributed adaptation coordination where multiple devices are available. GAS aims to provide a framework which assists ordinary users in reconfiguring or shaping their environments, like Lego blocks [7]. It is possible users to shape their environments as needed, however, it is too intrusive to force users to configure their environment when context changes occurs. It still works in progress to support dynamic adaptations and mappings among Plugs which provide transparent application reconfiguration.

3. Key Considerations

To enable a ubiquitous computing environment, it is essential that systems or applications not only aware the current context of the environment but also ask minimal involvement of users to reconfigure the system, adapting to the context. To meet the requirements, we focus on group-awareness as well as individual context, and transparent application reconfiguration with minimal user involvement.

3.1 Group-Awareness

Context-awareness enables a ubiquitous computing system to adapt to users proactively without distracting them. The existing context-aware applications are totally independent of each other and every person has different preferences, makes different decisions, and interprets differently in the same context. It is possible for one user's context adaptation to interfere others' interest by accident in a shared environment. Thus conflicts among them occur. It is unreasonable to provide services to a single person by sacrificing others. Thus, we need to not only consider the contexts of individual user but also put them together into group context, a set of context of individual users. In fact, group context is more than the context of a collection of individual users. We take into account contexts of individual users as group context in terms of conflicts among them, harmonization, and a new context comes out only when users are together as a group. In this paper, we focus on the conflict of contexts because it is a critical issue in the shared environment where a set of personalized context-aware applications run for different purposes independently.

3.2 Transparent Application Reconfiguration

Ubiquitous computing environments aim to provide nomadic users to access and manipulate information anywhere and anytime without directly dealing with a computer and/or a network. When a user moves to a new environment, his applications have to adapt according to changes according to the resources available in the environment and provide mechanisms to automatically adapt their preferences with no or minimal involvement of a user.

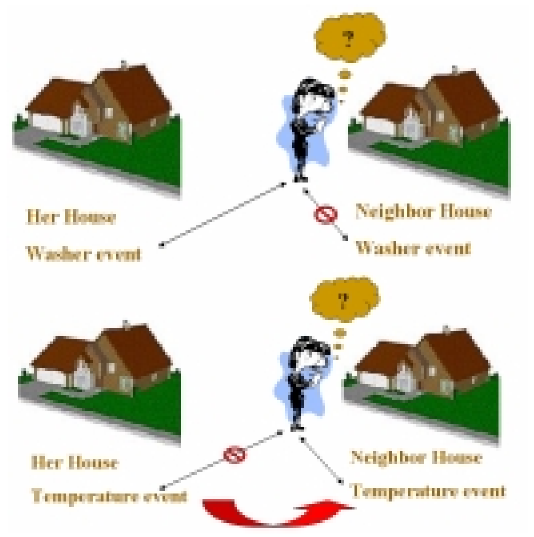


Fig. 1 Seamless Mobility Support

For instance, if a housewife wants to know washer information about her house in the neighbor's house with her handheld device, the washer information should be filtered between the same information generated from the two houses without her explicitly notifying her intents. We call it as anchored subscription. Another example is that she activates a healthcare application, which notifies temperature information because her disease is sensitive to temperature, and visits neighborhood with her handheld device. The healthcare application should automatically change information channel to receive temperature information from thermometer service run on the neighborhood without user interventions. We call it as localized subscription. Fig. 1 shows the situations respectively.

4. Proposed Architecture

A ubiquitous computing environment can be defined as a place or environment where people can do their works without considerations that computers or devices exist. To support such an environment, following functionality is essential: system should 1) sense environment changes and 2) provide mechanisms to gather the sensed information. The raw information should 3) be converted to context information. With the context information, 4) decision should be made itself to provide efficient services to users without intrusion to the users. However, all the services chosen by the decision engine may not be available because environment is seamlessly changing over time. 5) Service discovery mechanism is also essential to discover currently available service based on current context. New services or devices should 6) be dynamically plugged into system without intervention of administrator or users. The new service adaptation may incur context changes in the environment. Thus, changed information should 7) be distributed to other entities in the environment.

To accomplish the considerations and basic requirements described above, our middleware is composed of four components: dynamic reconfiguration support, context management, environment sensing, and service discovery. Fig. 2 shows overall architecture of proposed system.

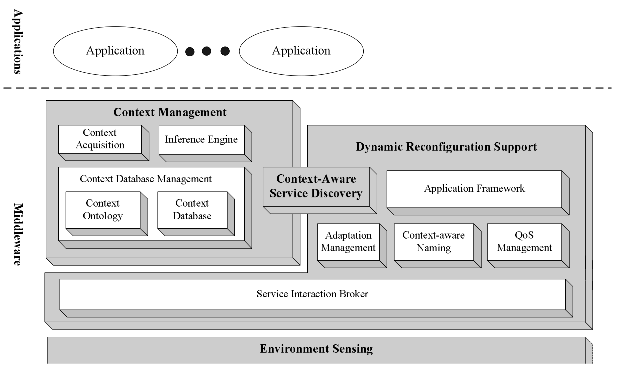


Fig. 2 Overall Architecture

4.1 Context Management

Context-awareness enables a ubiquitous computing system to adapt to users proactively without distracting them. In many research works [3], [17], contexts are in forms of digitized data taken from various types of sensors. It represents some features of a physical environment such as humidity, a sound level, and so on. Some other approaches define contexts which include non-sensory inputs such as the role of a user, application running on the computer, list of websites that have been visited, and the daily schedule. Context-aware applications interpret the contexts collected from various sources, and then, deliver services or adapt themselves to what a user wants to make them in given contexts in advance.

However, since the existing context-aware applications are totally independent of each other, it may intrude others' interest by accident in a shared environment where conflicts among them frequently occur. Thus we need to not only consider the contexts of individual user but also put them together into a group-context, a set of context of individual users. In fact, a group context is more than the context of a collection of individual users. We take into account contexts of individual users as group-contexts in terms of conflicts among them, harmonization, and a new context comes out only when users are together as a group.

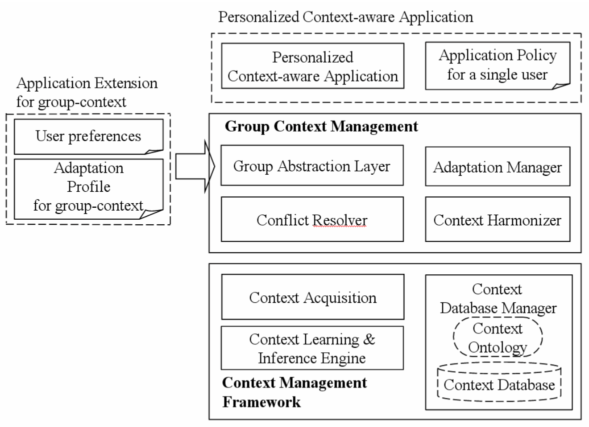

Personalized Context-aware Application
Personalized Context-aware Application
Application Policy for a single user
Application Extension for group-context
User preferences
Adaptation Profile for group-context
Group Context Management
Group Abstraction Layer
Adaptation Manager
Conflict Resolver
Context Harmonizer
Context Acquisition
Context Learning & Inference Engine
Context Management Framework
Context Database Manager
Context Ontology
Context Database

Fig. 3 Context Management Component Architecture

In a group-aware environment, the context management coordinates context-aware applications so that they do not sacrifice others unintentionally to achieve their goals. To enable group-awareness in ubiquitous computing environment, we define three components on top of fundamental components for context management, which are defined in other research prototype systems [3], [15]. As shown in Fig. 3, they are Group Abstraction Layer, Conflict Resolver, Adaptation Manager, and Context Harmonizer. The Context Acquisition component is also one of fundamental components in context management. In our approach, it has a role of detecting conflict and harmonization between context-aware applications as well as identifying user's intention from contexts. To detect a conflict, we introduce two dedicated user preferences, intention and desire. The intention represents is an abstract service type that a user wants to take when he is in an activity, e.g., turn lights on when she opens a door. The desire represents a list of abstract service types that a user does not want to take when she in a state, e.g., turn lights on when she sleeps in a bed. The Context Acquisition detects a conflict a user's intention matches the other's desire in the same context. The Group Abstraction Layer hides the underlying details of group context management from context-aware applications. The applications do not know about a conflict in a way that the Group Abstraction Layer intercepts the notification of conflict detection and provides the adaptation profile with the Conflict Resolver and the Context Harmonizer. The Conflict Resolver and Context Harmonizer make a decision how to adapt both applications which participate in the group-context based on the adaptation pro-file. Details of conflict management are described in [13].

4.2 Dynamic Reconfiguration Support

Ubiquitous computing environments inherently compose of heterogeneous devices and services. Currently available services or resource could not be available in the near future because context is seamlessly changing over time. System also should be evolving over time in order to incorporate those changes. However, it is impossible to predict and prepare all available services or resources before system starts up. To support such an environment, Dynamic Reconfiguration Support aims to provide polymorphic service composition which dynamically changes applications' form to fully leverage current available resources.

Dynamic Reconfiguration Support is composed of mainly three component, service interaction broker, application framework, and adaptation support as shown in Fig. 4. Service interaction broker not only provides common communication facilities among objects but also support seamless application mobility. We use publish/subscribe event model as communication model because it gives flexibility by decoupling event sources from sinks and binding them as needed at runtime. Application framework component provides application model and interfaces to application developers and gives clear separation between application and middleware.

To support seamless application mobility (anchored and localized subscription described in section 3.2, we exploit two concepts, *link objects* and *sub-typing*. Link objects hold event subscription information of an application as an object. When a user moves to another environment, the link object is passed in order to rebind application with the information. With the link object, it is possible for a target application to be transparently bound with locally available services without direct involvement of the application such as parsing application script information. Details of link object concept are described in [10]. When a user moves to a new environment, it is not always possible to assume that all the service objects or resources required by the user are available. To overcome this, we propose the sub-typing concept. Suppose that the event type *a'* is a sub-type of the event type *a* and object *A* wishes to subscribe to the event type *a'*. If only the event type *a* is available in the environment, the application is bound with the event type *a*, instead of event type *a'*. This allows applications to be dynamically bound with available services by exploiting the key concept of sub-typing, principle of substitutability [11]. Details of subtyping based adaptation mechanism are described in [9].

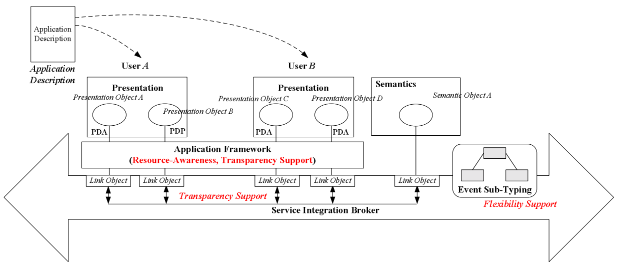


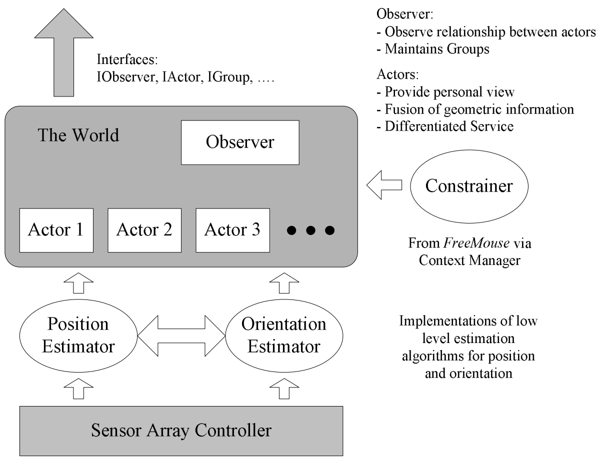
Fig. 4 Dynamic Reconfiguration Support Architecture

Dynamic Reconfiguration Support intimately cooperates with Context Manager and Context-Aware Service Discovery. For example, context change decided by Context Management may trigger new service adaptation. Dynamic Reconfiguration Support uses Context-Aware Service Discovery to search currently available services to be suitable to the context change. To adopt new service, it should check and verify avail-able resources or resource conflicts among services to avoid service crush or malfunctioning of applications.

4.3 Environment Sensing

One of the most frequently quoted keywords that characterize ubiquitous computing is invisibility; computers in ubiquitous computing should not wait for an explicit command from the user but should go out to monitor the current state of the user and the environment and make an active decision. The minimal list of environment variables that computers need to keep track of will be the physical state of the user (position, orientation, and so on) and the state variables of the environment (temperature, humidity, illumination, and so on). Among other types of environment data, the current de-sign of Environment Sensing is mainly focused on the positioning of the user, i.e., tracking of the position and orientation of the user using RF array technology. It shares the basic operating principle with RF intensity based positioning systems such as RADAR, but is designed to provide a resolution high enough to support ubiquitous computing applications at home. Moreover, it aims at providing both position and orientation information, which is not very common in RF-intensity based systems.

The sensing module consists of an array of RF-intensity sensing elements and an array controller that keeps array elements in synchrony. RF signals from trackers worn by the users and appliances are periodically collected by the controller and will be sent to the signal processing subsystem. The two main components of the signal processing subsystem are Position Estimator and Orientation Estimator. Both modules utilize a kind of curve-fitting algorithm that minimizes errors between observed intensity values and reference intensity data. Raw estimation data from the signal processing subsystem will be pushed to the position database subsystem that consists of many Actor objects and one Observer object. An Actor representing an object in the environment (a user or an appliance) keeps track of the physical state of the object. Its unique role, however, is to utilized constraint relation with other Actors in order to refine the raw position information provided by the signal processing module. Also, it provides an interface that gives the position information of other object from its own perspective. The Observer keeps track of relative position information of Actors and maintains virtual actors which represent groups. The primary consumer of the position information in the current framework will be Context Manager. Fig. 5 shows the location tracking component in Environment Sensing.



Interfaces:
IObserver, IActor, IGroup, ….
Observer:
- Observe relationship between actors
- Maintains Groups
Actors:
- Provide personal view
- Fusion of geometric information
- Differentiated Service
The World
Observer
Actor 1
Actor 2
Actor 3
Constrainer
From FreeMouse via Context Manager
Position Estimator
Orientation Estimator
Implementations of low level estimation algorithms for position and orientation
Sensor Array Controller

Fig. 5 Location Tracking Component Architecture

4.4 Context-Aware Service Discovery

We expect that there will be so many same/similar service instances in ubiquitous computing environments. For example, multiple TVs or other display devices will be equipped in the future home. However, traditional service discovery protocols [22], [23], [24], [25] which use property based matching mechanisms, can not provide filtering functionality enough for applications to select optimal service instance, that is, they can't decide which service instance is most relevant to the application's current situation. For selecting most appropriate one among multiple same service instances, service discovery should be aware of context information of applications. Service instances are evaluated based on the extent of fitness to current context such as current location, or preferences.

To achieve this, Context-Aware Service Discovery needs the following components; Context Constraints, Context Information Extractor, and Service Evaluator. To evaluate service fitness, Context-Aware Service discovery should know which context in-formation is relevant to the required service and how to evaluate fitness of service instance to relevant context. We call this information as Context Constraints. Next, Context-Aware Service Discovery should get needed context information value ac-cording to the context constraints. Context Information Extractor provides Service Evaluator with context value needed for service evaluation. Service Evaluator evaluates discovered service instances using evaluation rule embedded in context constraints and context value provided by Context Information Extractor. Finally, most relevant service to the context constraints is returned to the service discovery requestor. Fig. 6 shows that the overall architecture of Context-Aware Service Discovery Component.

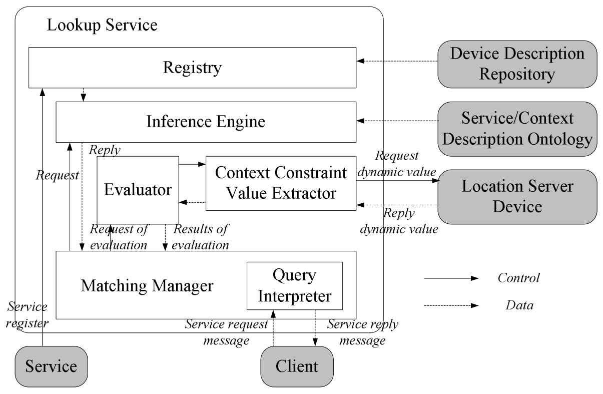
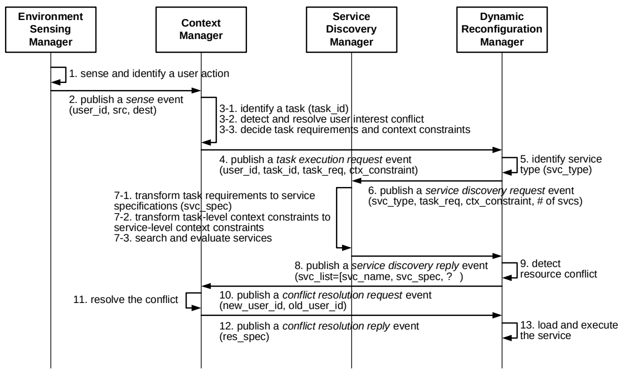


Fig. 6 Context-Aware Service Discovery Architecture

5. Prototype Implementation

The prototype system consists of five major components: Service Interaction Broker, Context Manager, Dynamic Reconfiguration Manager, Context-Aware Service Discovery Manager, and Environment Sensing Manager. Service Interaction Broker enables other components communicate with each other by publishing and subscribing events. A component wanting to publish an event, creates an instance of ASEvent class, sets some parameters, and publishes it to the Service Interaction Broker by calling publish() method in ASEventManager class. On the other hand, a component, which wants to receives events of a specific type, should implement ASEventHandler interface and subscribe to a specific type of events by calling subscribe() method in ASEventManager class. When an event is published by some components, Service Interaction Broker delivers the event to other components that have subscribed to the type of the published event and notifies by calling handleEvent() method of the components. Dynamic reconfiguration manager takes the responsibility for performing a task that is composed of several services. It maintains the task-to-service mappings to determine the required services to perform a task. It also monitors the resource usages and then detects a resource conflict. Context manager manages the context information in the ubiquitous computing environment. It receives environment sensing information from environment sensing manager (various types of sensors or input devices) and transforms the raw information to the context information. Then, it identifies the task that should be performed from the context information. It also is responsible for resolving the conflicts between users. Service discovery manager maintains the list of services available in an environment. When receiving a search request from dynamic reconfiguration manager, it finds the proper services that match some service specification and context constraints and replies the information to dynamic reconfiguration manager. Environment sensing manager now consists of two types of sensors: the first is RF sensor and the second is Free Mouse. RF sensors detect some user actions and send that information to context manager. Free Mouse, which acts as input devices of a user, also detect user actions and send that information to context manager. Fig. 7 shows the interaction diagram among components.



Environment Sensing Manager
Context Manager
Service Discovery Manager
Dynamic Reconfiguration Manager
1. sense and identify a user action
2. publish a sense event (user_id, src, dest)
3-1. identify a task (task_id)
3-2. detect and resolve user interest conflict
3-3. decide task requirements and context constraints
4. publish a task execution request event (user_id, task_id, task_req, ctx_constraint)
5. identify service type (svc_type)
6. publish a service discovery request event (svc_type, task_req, ctx_constraint, # of svcs)
7-1. transform task requirements to service specifications (svc_spec)
7-2. transform task-level context constraints to service-level context constraints
7-3. search and evaluate services
8. publish a service discovery reply event (svc_list=[svc_name, svc_spec, ?])
9. detect resource conflict
10. publish a conflict resolution request event (new_user_id, old_user_id)
11. resolve the conflict
12. publish a conflict resolution reply event (res_spec)
13. load and execute the service

Fig. 7 Brief Interaction Diagram among Components

With prototyped middleware, we build ubiquitous home environment, called *Digital Butler*, as shown in Fig. 8, which primarily aims to minimize administration overheads of residents and to provide personalized service with internet-enabled appliances for near-future home environment.

Fig. 8 Prototyped Digital Butler

5. Concluding Remarks

In a ubiquitous computing environment, computers and networks become part of the environment and not directly visible unlike today. In this paper, we propose Active Surroundings, a middleware infrastructure for a ubiquitous computing environment where entities such as services and devices actively response to user actions or help users to perform their jobs according to the current context of users. We focus on group-awareness and transparent and seamless application reconfiguration. For these goals, our middleware is composed of four main components such as Environment Sensing, Context Management, Context-Aware Service Discovery, and Dynamic Reconfiguration Support to meet the requirements of ubiquitous computing environments. The proposed middleware allows applications to handle runtime detection and resolution of context conflicts among group members. To detect a conflict, two dedicated user preferences, *intention* and *desire* are introduced. An intention is asserted in the system when a user is in a certain activity and a desire is asserted when in a state. A conflict is detected when the intention of a user is matched with the desire of the other. We introduce the *link object* and exploit *sub-typing* to support transparent dynamic reconfiguration. With the link object, an application is bound once to the require service types via link objects. Even when the user context changes due to user mobility, only the link object is reconfigured with appropriate service in a new environment and the application does not need to be involved. Furthermore, when the required service is not available in the new environment, its super-type instance is sought instead and bound if exists. The sub-typing based event subscription gives flexibility in heterogeneous environments.

We have prototyped the proposed middleware and currently focus on application framework which leverages presentation/semantic split application model as well as context-aware naming which aims to provide high-level service name transparency. We are revising and experimenting our conflict detection and resolution method with our prototype to improve its effectiveness. We also work on high-level context modeling and representation with the Semantic Web ontology for inferring group context from a collection of individual contexts.

References

1. Capra L., Emmerich W., and Mascolo C. “CARISMA: Context-Aware Reflective Middleware System for Mobile Applications,” IEEE Trans. on Software Engineering, Vol. 29. (2003) 929-945.
2. Chen H. “An Intelligent Broker Architecture for Context-Aware Systems,” PhD. dissertation proposal, the University of Maryland Baltimore County. (2003)
3. Chen G., and Kotz D. “Solar: A pervasive-computing infrastructure for context-aware mobile applications,” Dartmouth College Technical Report, TR2002-421, (2002).
4. Efstratiou, C., Friday, A., Davies, N., and Cheverst, K. “A Platform Supporting Coordinated Adaptation in Mobile Systems,” 4th IEEE Workshop on Mobile Computing Systems and Applications 20-21 June (WMCSA’02) 128 – 137.
5. Garlan, D., Siewiorek, D., Smailagic, A., and Steenkiste, P. “Project Aura: Towards Distraction-Free Pervasive Computing,” IEEE Pervasive Computing, Vol. 1, No. 2, (2002), 22-31.
6. Guttman, E., Perkins, C., Veizades, J., and Day, M. “SLP, Ver. 2,” http://www.rfceditor.org/rfc/rfc2608.txt, (1999).
7. Kameas, A., Bellis, S., Mavrommati, I., Delaney, K., Colley, M., and Pounds-Cornish, A. “An Architecture that Treats Everyday Objects as Communicating Tangible Components,” 1st IEEE International Conference on Pervasive Computing and Communications (PerCom’03), 23-26 March (2003) 115-122.
8. Kindberg, T., and Barton, J. “A Web-based nomadic computing system,” Computer Networks, 35, (2001), 443-456.
9. Lee, D., Han, S., Lee, K., and Kim, M. “A Behavioral Subtyping-based Application Adaptation Scheme for Ubiquitous Computing Environments”, Technical Report, TR-CS-20040730, available at http://cds.icu.ac.kr
10. Lee, K., Han, S. and Lee, D. “Service Interaction Broker for Ubiquitous Computing Environments,” Technical Report, TR-CS-20040820, available at http://cds.icu.ac.kr
11. Liskov, B., and Wing, J. “A behavioral notion of subtyping,” ACM Tran. Prog. Lang. and Systems, Nov. 1994.
12. Masuoka R., Labrou Y., Parsia B., and Sirin E. “Ontology-Enabled Pervasive Computing Applications,” IEEE Intelligent Systems, Vol. 18, (2003) 68-72.
13. Park, I., Hyun, S., and Lee, D., “Context-Conflict Management for Context-aware Applications in Group-aware Ubiquitous Computing Environments,” Technical Report, TR-CS-2004017, available at http://cds.icu.ac.kr
14. Rakotonirainy, A., Indulska, J., Loke, S., and Zaslavsky, A. “Middleware for Reactive Components: An Integrated Use of Context, Roles, and Event Based Coordination,” IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, (2001), 77-98.
15. Ranganathan A., Campbell R. “An Infrastructure for Context-Awareness based on First Order Logic,” Journal of Personal and Ubiquitous Computing, Vol. 7. (2003) 353-364.
16. Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., and Nahrstedt, K. “Gaia: A Middeware Infrastructure to Enable Active Spaces,” IEEE Pervasive Computing Magazine, Vol. 3. (2002) 74-83.
17. Van Laerhoven K., and Aidoo K. “Teaching Context to Applications,” Journal of Personal and Ubiquitous Computing, Vol. 5. Feb. (2001) 46-49

18. Weiser, M. “The Computer for the 21st Century,” Scientific American, 265(3), September (1991) 94-104
19. Yau, S., Karim, F., Wang, Y., Wang, B., and Gupta, S. “Reconfigurable Context-Sensitive Middleware for Pervasive Computing,” IEEE Pervasive Computing Magazine, Vol. 3, (2002), 33-40.
20. Zhao, W., Schulzrinne, H., Guttman, E., Bisdikian, C., and Jerome, W. “Select and Sort Extensions for the Service Location Protocol,” IETF RFC 3421, (2002)
21. Korpipaa P., Mantyjarvi J., Kela J., Keranen H., and Malm E. “Managing context information in mobile devices,” IEEE Pervasive Computing, Vol. 2, (2003) 42-51.
22. JINI, http://www.jini.org
23. Salutation, http://www.salutation.org
24. Semantic Web, http://www.w3.org/2001/sw/
25. UPnP, http://www.upnp.org