

Adapting Ubicomp Systems to a Common Model

Michael Blackstock¹, Rodger Lea², and Charles Krasic¹

¹ Department of Computer Science, University of British Columbia
201-2366 Main Mall, Vancouver, B.C., Canada

² Media and Graphics Interdisciplinary Centre, University of British Columbia
FSC 3640 - 2424 Main Mall, Vancouver, B.C., Canada
{michael@cs, rodgerl@ece, krasic@cs}.ubc.ca

1 Introduction

Until recently, system designers have focused primarily on supporting applications and user access to integrated ubiquitous computing environments or “smart spaces” within single administrative or network domains. We believe that the wider deployment of ubicomp systems will be more viable when applications can interact with devices, services and sensor information independent of their environment and across domain boundaries. Cross-domain integration and interoperability will rely on a shared understanding of the possible interactions between applications and environments: a common model for ubiquitous computing environments is required. In previous work [1, 2] we have designed such a model for ubicomp environments based on the following high level abstractions derived from a comprehensive survey of existing systems:

- **Environment Model** that encapsulates the current state of the environment as a whole.
- **Entities** such as people, places, things, groups and activities.
- **Context** associated with entities including values such as location, temperature, direction, or time.
- **Services** or functionality associated with entities.
- **Entity Relationships** such as geometric, social, and activity-related relationships
- **Events** that entities can fire related to a change of state of an entity, for example a door closing, or a temperature change.
- **Data or Content** related to an entity such as user’s personal notes, or documents associated with a meeting.

Our entity-centric model for ubicomp environments is called the Ubicomp Common Model. Entities such as people, places, and things, hosted by the environment are related to context, services, events and data they support. To evaluate this model we have begun work on a meta-middleware platform called the Ubicomp Integration Framework (UIF) designed to adapt existing systems’ abstractions to the UCM. Through this integration work we intend to gain an understanding of common patterns used by existing systems and assess the complexity of mapping these systems to the

UCM. This position paper highlights several integration challenges uncovered during our ongoing evaluation work designing and implementing integration adapters.

2 Integration Adapter Challenges

The responsibility of an adapter in the UIF system is to convert the UCM abstractions and associated API to and from native middleware APIs as shown in Figure 1. More specifically an adapter is responsible for mapping entity ids, context attribute names, marshalling context values, service parameters, event data and subscriptions, and relaying relevant model changes between native middleware and UCM. An adapter also provides the UIF with information about the current relevant components and their capabilities.

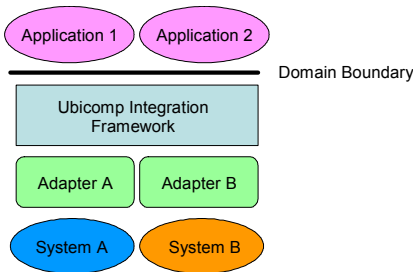


Fig. 1. Adapter A and B integrate ubiquitous computing system A and B to the Ubicomp Common Model.

While much of an adapter’s functionality is clear, other responsibilities of an adapter are more challenging. In particular, an abstraction supported by the UCM may not be directly supported by an underlying system. In this case either the integrator must provide missing information to the system, an adapter implementation must compensate for this missing abstraction, or a combination of both. We discuss three examples of this in the following subsections.

2.1 Environment Model from Component Attributes

Several component-oriented ubicomp middleware systems such as the Context Toolkit (CTK) [3] or the Equip Component Toolkit (ECT) [4] do not support an explicit model containing the relevant entities (people, places, things, groups) that in the environment. The Context Toolkit’s *Discoverer* component for example, only holds information about the components available in the environment. This allows applications to find and use components such as *context widgets* based on various attributes. Depending on its implementation, a widget may or may not supply explicit information about the entity that its supplied context applies to. For example, a constant location attribute for a lighting widget may specify the location it applies to. However,

whether this attribute exists, what it is called and what value it returns is component implementation dependent.

To compensate for the lack of an explicit entity-centric model, the UIF system can start with a model description supplied by an integrator [5]. This description provides information about the static elements of the model such as buildings, rooms, and fixed objects such as tables, printers and their spatial relationships. It can also supply information about the supported users in that environment. The UIF can then incorporate information about components supplied by an adapter at run time. The UIF can relate components to entity information supplied by an integrator based on the constant and dynamic attribute values supplied by a widget, and provide information about new entities such as people and places based on attribute values.

For example, one can infer the existence of the MAGIC Lab entity, when a component contains the location attribute value “MAGIC Lab”. Similarly the system can infer the existence of a user entity in the environment model when a location widget supplies that user’s name. Overall though, maintaining an explicit entity-centric model where none exists is a challenge.

2.2 Inferring Entity Relationships

A common abstraction used by systems such as Cooltown [6], EasyLiving [7], and others [8, 9] is *entity relationships*. Entity relationships are a form of context that relate one entity to others. These relationships may be special location-based such as *contained-in* or social relationships such as *friend-of*.

When entity relationships are not supported by the underlying system they must be inferred in some way. For example, to determine the users in a room, it would seem natural to query a “containedIn” component attribute or property. One would expect an unordered list of user and other entity identifiers to be returned. In many cases, the underlying system or a specific component may not be designed to support this type of query. A presence component of an existing system may only fire events when a user enters or leaves a room for example. To support a more natural query for entity relationships, either the adapter or integration system must implement this relationship abstraction, for example, maintaining the current list of users contained in a room based on events received by the underlying component.

2.3 Other Abstraction Mismatches

During our evaluation we have found other abstraction mismatches that may be addressed by an adapter. An adapter may provide simple context interpretation [3] facilities such as mapping RFID device identifiers to user ids in associated context queries and events. In some cases an adapter may need to provide support for events when the underlying system only supports polling, or fire single higher level events triggered by multiple lower level events in the native environment. Depending on the Environment Profile [2] an adapter may present *content* as either a stand alone entity (such as a document) or as context (an *image* attribute of a user entity for example).

3. Conclusions

This position paper presents one integration challenge toward integrating several ubi-comp systems under a common environment model: the need to providing higher level abstractions on top of underlying systems where no such abstractions exist. We believe that developing approaches to addressing this issue will not only inform the development of the UbiComp Common Model, but also highlight common patterns used in existing systems and contribute to the design and implementation of future ubi-comp infrastructures.

References

1. Blackstock, M., Lea, R., Krasic, C.: Toward a Shared Model for Wide Area Interoperability of Ubiquitous Computing Environments. System Support for Ubiquitous Computing (UBISYS) 2006 Workshop at UbiComp, Newport Beach, CA (2006)
2. Blackstock, M., Lea, R., Krasic, C.: Toward Wide Area Interaction with Ubiquitous Computing Environments. 1st European Conference on Smart Sensing and Context (EuroSSC), Enschede, The Netherlands (2006)
3. Dey, A.K.: Providing Architectural Support for Building Context-Aware Applications. PhD Thesis. College of Computing, Georgia Institute of Technology (2000)
4. Greenhalgh, C., Izadi, S., Mathrick, J., Humble, J., Taylor, I.: ECT: a toolkit to support rapid construction of ubi-comp environments. Proceedings of UbiComp '04 (Demonstration). Springer, Nottingham (2004)
5. Blackstock, M., Lea, R., Krasic, C.: Managing an Integrated UbiComp Environment using Ontologies and Reasoning. 4th IEEE Workshop on Context Management and Reasoning (CoMoRea) 2007 at the 5th IEEE International Conference on Pervasive Computing and Communications, New York (2007)
6. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B.: People, places things: Web presence for the real world. Third IEEE Workshop on Mobile Computing Systems and Applications Monterey, California (2000)
7. Brumitt, B., Meyers, B., Krumm, J., Kern, A., Shafer, S.A.: EasyLiving: Technologies for Intelligent Environments. Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing. Springer-Verlag, Bristol, UK (2000)
8. Jonsson, M.: Context Shadow: An Infrastructure for Context Aware Computing. Workshop on Artificial Intelligence in Mobile Systems (AIMS), Lyon, France (2002)
9. Bardram, J.E.: The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. Pervasive Computing: Third International Conference. Springer Berlin / Heidelberg, Munich, Germany (2005)