Managing an Integrated Ubicomp Environment using Ontologies and Reasoning

Michael Blackstock¹, Rodger Lea², Charles Krasic¹

¹Computer Science Department, ²Media and Graphics Interdisciplinary Center, University of British Columbia, Vancouver, Canada
{michael@cs, rodgerl@ece, krasic@cs}.ubc.ca

Abstract

One issue hindering the deployment of integrated ubicomp environments is the lack of a shared model for applications to utilize resources across administrative and network domains. Through the use of concrete examples, we describe our proposed common model for ubicomp environments to address this issue. We then present use cases and additional examples for how our Ubicomp Common Model (UCM) is used by our integration framework. The Ubicomp Integration Framework (UIF) uses an Ontology Driven Architecture (ODA) [2] and Enterprise Application Integration (EAI) techniques to integrate existing ubiquitous computing middleware to the common model. The use cases illustrate how the integrated model and reasoning is used to maintain a dynamic integrated model of an environment.

1. Introduction

Despite many years of research, the deployment of ubiquitous computing (ubicomp) systems to support “smart spaces” or integrated ubicomp environments has not been widespread. One issue hindering deployment is the lack of a shared model for these environments for greater application reuse between domains. This is needed for mobile devices applications to access local resources in public places such as museums and shopping malls, or when composing ubicomp environments such as the home and office. While a common model is desirable for interoperability, it is likely too early to agree on the same programming abstractions for all environments and classes of applications. To address this, we have proposed the Ubicomp Integration Framework (UIF) to expose a common interoperable environment model to applications independent of the underlying middleware used [1]. The framework design borrows techniques from enterprise application integration [EAI] architectures to expose existing ubicomp systems to applications outside of the administrative and network domains.

Recently, researchers have suggested the use of Ontology Driven Architectures [2, 3] to ease the development and management of applications hosted by enterprise middleware systems. We make use of these methodologies and employ an executable model in the UIF to assist in the configuration and run time maintenance of an integrated ubicomp environment. An ontology called the Ubicomp Common Model (UCM) is used to specify integration-time configuration information of our platform, made up of both the exposed ubicomp environment abstractions accessible by applications and the components used to implement these abstractions. A reasoning engine maintains both the exposed conceptual model and implementation at run time, dynamically adjusting the model as the environment changes.

This paper describes the Ubicomp Common Model through the use of examples. The main contribution is a list of use cases for how the model is used for both static configuration and addressing the challenges related to dynamic run time maintenance of the model by our system.

The rest of this paper is organized as follows. In Section 2 we provide some background and compare the UIF to related work. In Section 3 we provide a high level overview of the UIF system. In section 4 we describe the UCM. In Section 5 we provide use cases for our model in the context of its use in our integration middleware. We conclude with some open questions for discussion at the workshop.

2. Background

In the Object Management Group’s Model Driven Architecture (MDA) initiative, formal models are used to generate artifacts for downstream use in the development of software systems [4]. Domain specific conceptual models are used during the design phases that are independent of the system implementation. These are combined with specifications of the platform to generate code for a specific platform such as J2EE, CORBA or Web Services.
More recently researchers have begun to explore the potential uses of Semantic Web technologies in software engineering to extend the MDA framework. These include the use of ontologies as formal model specifications, and the use of semantic web storage and reasoning technologies for storing, querying and reasoning about runtime information. It is believed that semantic models, or ontologies, will augment the MDA standards and methodologies, giving rise to Ontology Driven Architectures (ODA) [2].

Figure 1. Ubicomp Integration Framework

One application of ODA is to describe the components and their relationships in a system. The KAON server [3] for example, uses ontologies to provide an explicit, executable model of J2EE application server components and other abstractions within an enterprise application server. This model is used to query and reason about component dependencies, security and other information to perform tasks at development and run time. Similarly, we’ve found that a conceptual model of ubicomp environments such as information about physical entities and associated context and service interfaces can also be formalized and maintained in an integrated model. This model forms the basis for maintaining system behavior implemented by the dynamic composition of various components.

Several context-aware systems have used ontologies to describe context and relationships between entities [5, 6]. Others have used ontologies to address interoperability between devices in an environment [7]. The Ontology Server integrated with the Gaia system [8] used ontologies to support configuration management, service discovery, human interfaces, assist in the interoperability between various software components and perform context inference within Gaia. Unlike these, the UCM ontology relates exposed abstractions such as context, events and services to implementation components hosted by existing ubicomp middleware.

3. UIF Overview

The goal of the UIF is to adapt existing ubicomp middleware systems to a common model for wide area interoperability. Like the KAON application server, an executable model is tightly integrated into the UIF itself. In contrast to KAON, the model and reasoning is used not only for component specifications and management, but also to describe an implementation-independent ubicomp environment model for applications.

A typical deployment of a UIF system and integrated is shown in Figure 1. The UIF is used to integrate one or more existing ubicomp systems (iROS [9], Context Toolkit [10] and Equator Component Toolkit [11] shown here as an example) under a single integrated environment model accessed using web services. When an application calls the UIF to retrieve context or invoke a service, the Web Services Façade delegate this call to the Environment Composition Logic. Queries are performed on the current model to find the components, and adapters that implement services and supply context and events. Method calls are then delegated to the native components or ubicomp middleware adapters. To maintain entity relationships and perform context interpretation, a Reasoner subsystem polls context sources for relevant updated context information. The context information retrieved is converted to Web Ontology Language (OWL) [12] statements and added to the model. This allows the system to apply rules that use current context values for maintaining the model.

Figure 2. Ubicomp Common Model Aspects

4. Model Design and Examples

The UCM design is based on taxonomy of abstractions derived from a comprehensive survey of existing ubicomp systems [1]. It consists of three related aspects of a ubicomp environment as shown in Figure 2: the exposed physical entities (people, places and things), their context and relationships called the Environment State, meta-data related to the services
and context associated with these entities, called the *Environment Meta-State*, and the components used to implement the environment called the *Implementation*. These related aspects of the ubicomp environment are managed by the knowledge base of the UIF supporting system. In the following subsections we describe the three aspects of our model design through the use of examples.

### 4.1. Environment State

The following OWL statements describe a `coffeeShop` entity. It has some static context associated with it using the location attribute in the `ucm` namespace specifying its GPS coordinates using the `campusLocation` context value. This fragment also establishes the entity-entity relationship between the `coffeeShop` and the entire campus, called `campusPlace` in the `campus` namespace.

```xml
<campus:CampusBuilding rdf:ID="coffeeShop">
  <ucm:location>
    <ucm:Position rdf:ID="campusLocation">
      <ucm:latitude>-50.2335</ucm:latitude>
      <ucm:longitude>124.44555</ucm:longitude>
    </ucm:Position>
  </ucm:location>
  <ucm:containedIn rdf:resource="&campus;campusPlace"/>
</campus:CampusPlace>
```

### 4.2. Environment Meta State

It is important for applications to determine the context types, service interfaces, and event types that are associated with entities in an environment. With this information, applications can query the environment to see if its requirements can be satisfied. The following fragment associates the context type `pointLocation` with a `CampusUser` entity called `bob`. The `bob` entity can also fire the event type `contextChangeEvent`, and exposes the `MessageService` interface.

```xml
<campus:CampusUser rdf:ID="bob">
  <rdfs:label>Bob Smith</rdfs:label>
  <ucm:containedIn rdf:resource="&campus;campusPlace"/>
  <ucm:hasContextType rdf:resource="&ucm;pointLocation"/>
  <ucm:firesEventType rdf:resource="&ucm;contextChangedEvent"/>
  <ucm:exposes rdf:resource="&campus;MessageService"/>
</campus:CampusUser>
```

The following fragments declare the `ContextType`, `EventType` and `ServiceInterface` associated with the `bob` entity above.

```xml
<ContextType rdf:ID="pointLocation">
  <usesContextAttribute rdf:resource="#location"/>
</ContextType>

<EventType rdf:ID="contextChangedEvent">
  <fires rdf:resource="#contextChanged"/>
</EventType>

<ucm:ServiceInterface rdf:ID="MessageService">
  <rdfs:label>MessageService</rdfs:label>
</ucm:ServiceInterface>
```

### 4.3. Implementation

Existing ubicomp systems provide a homogenous layer on top of context sensors, actuators, and software services. Similarly, the UCM describes `ContextSource`, `EventSource` and `Service` component types that encapsulate components either hosted by the UIF or by an external ubicomp system connected by an Adapter.

The following OWL fragment describes a context source component that provides locations of users on a campus. The component is called `positionSource` and it is of type `CampusPositionSource`, which is a subclass of `ContextSource`. This component is implemented (binding property) by a Java class hosted by the UIF platform itself. The `onStartup` property indicates that this component should be instantiated when the system starts. The `hasContextType` property indicates to the system that this component supplies a `pointLocation`. Similarly it supplies a `contextChangedEvent`.

```xml
<campus:CampusPositionSource rdf:ID="positionSource">
  <ucm:binding>
    ca.ubc.cs.uif.prototype PositionContextSource
  </ucm:binding>
  <ucm:hasContextType rdf:resource="#pointLocation"/>
  <ucm:firesEventType rdf:resource="#contextChangedEvent"/>
  <ucm:exposes rdf:resource="#campus;MessageService"/>
</campus:CampusPositionSource>
```
4.4. Linking Exposed State to Implementation

The exposed aspects of the environment are linked with the implementation using the aggregates relationship. When an entity aggregates a component, it then exposes the context, events and service interfaces implemented by that component. These axioms are supported by rules like the following:

\[
\text{(entity ucm:hasContextType ?contextType)} \iff \\
\text{(entity ucm:aggregates ?component)} \wedge \\
\text{(component ucm:hasContextType ?contextType)} \wedge \\
\text{(component rdf:type ucm:Component)}
\]

The rule states that if an entity aggregates a component, and a component has a context type \text{contextType}, that entity also has that \text{contextType}. A similar rule applies to events and services.

5. Use Cases

In this section we provide some use cases for the use of our UCM ontology and reasoning in the UIF. While the benefits of the integrated model are mainly realized through run time use cases, design and integration time use cases are presented here for completeness.

5.1. Design/Integration Time Use Cases

At environment design time, ontologies can be used to describe the initial state and static aspects of the environment model.

**Static environment model descriptions.** In any environment there are static entities such as rooms, furniture, and fixed devices. Ontologies can be used to describe these entities and static context such as the location of fixed objects. Important static relationships such as relative location can also be described. For example, a printer is containedIn a room, and a room is containedIn a building.

**Interface semantics and context quality.** Even when a component exposes the same interface, the behavior or quality of its service may vary from one implementation to another. A GPS location sensor may provide location accuracy of 5m outdoors whereas an indoor location sensor may provide only room level accuracy indoors for example. Semantic descriptions can make interfaces explicit to applications so that they know when they can and should be applied.

**Component configuration information.** In the UIF, components in the system may be implemented by native Java classes hosted by the UIF or by integrated ubicomp middleware. In the UIF, the \text{binding} property is used to associate a component in the model with a Java class implementation; the \text{adapter} property associates the component with integrated middleware.

**Initial entity-component dependencies.** A physical entity such as a device will often have services, context or events associated with it. The aggregation relationship between entities and implementation components establishes the initial link between the conceptual or exposed model of an environment and its implementation components.

5.2. Run time use cases

A challenge for any ubicomp system is in managing the dynamic aspects of an environment for applications. In our system, ontologies and reasoning can be used to maintain the exposed model and associated components as their composition changes. We have found these technologies to be useful in the following cases:

**Entity Composition.** When new entities such as mobile users and their environment arrive and leave the environment, associated information about these entities can be added to the model. The reasoning engine can then associate these new entities with the appropriate components, types and service interfaces. In our system, an \text{addEntity()} web service method adds an entity and associated statements to the integration model. Similarly, the \text{removeEntity()} call removes the entity and associated statements.

**Context interpretation.** Ontology based systems such [5, 6] have shown that it is possible to use an integrated reasoning engine to infer higher level context from lower level context information. Cooltown [13] and other systems have demonstrated the value of entity relationships for service discovery and other use cases. A reasoning engine with appropriate rules can be used to infer such relationships from lower level context information. For example, the following rule establishes the \text{near} relationship when one entity is within 50 metres of another.

\[
\text{(entity1 ucm:near ?entity2) \iff} \\
\text{(entity1 ucm:location ?position1)} \\
\text{(entity2 ucm:location ?position2)} \\
\text{notEqual(entity1, entity2)} \\
\text{inRange(position1, position2, 50)}
\]

**Dynamic component call/message routing.** At run time, the model can be queried to determine the component that currently supplies the requested service or context which may depend on the current situation. For example, one context source may be used to determine a user’s location indoors and another outdoors. The following rule will aggregate a context
source \( (\text{csIndoorPositionSource}) \) for providing indoor position information only when a \text{Person} entity is containedIn the \text{csBuilding} place.

\[
[\{?\text{entity ucm:aggregates campus:csIndoorPositionSource} \} <-
\{?\text{entity rdf:type ucm:Person}\}
\{?\text{entity ucm:containedIn campus:csBuilding}\}]
\]

**Entity classification and discovery.** A client of the system or internal component may need to find the entities in the environment that match a certain criteria such as its type or the service interfaces or context types they provide. Similarly clients or components may need to know the interfaces or context supported by a given entity. For example, In our system, a web service method called, `findEntities()` allows the client to find entities using the `WHERE` clause of a SPARQL query.

**Situation events.** With an integrated model of an environment, it is possible to query or be notified when a certain situation exists in the environment as a whole. For example, one can query whether there is more than one user entity contained in a meeting room. This may be used by an application to infer that a meeting may be starting to call appropriate services.

**Security.** We believe that an executable model can be used to express roles and permissions of entities and accessing applications. At run time when an application calls a service or requests context, the model can be queried to apply active security policies.

### 6. Conclusions and Open Questions

In the design of our UIF system, we have applied an Ontology Driven Architecture that makes use of Semantic Web technologies to integrate existing ubicomp middleware to a common model (UCM) for interoperability. We have described the three aspects of the UCM through the use of examples, and described several use cases for Semantic Web technologies at integration and run time. Based on our initial prototype work, this approach seems promising, however, several open questions remain that we hope to discuss at this workshop and address in future work:

- Can the exposed aspects of our environment model support a wide range of applications? Have we captured the right abstractions for the resources exposed by existing ubicomp systems in the implementation aspect of our model?

- Should the clients or applications of an environment be explicitly expressed in our model? This may allow us to use the model for reasoning about security and access device capabilities.

- The UIF system requires maintaining an aggregated model of the environment that is in sync with the models exposed by the adapted systems. How can we do this in an efficient and scalable manner?

### 7. References


