Ubiquitous Computing
University of Illinois at Urbana-Champaign
Computer Science Department
Request for Comments: 0002
Obsoletes: none

Anand Ranganathan
Software Research Group
September 2001

The Space Repository

Table of Contents

1. Status of this Memo

This memo is in the Request for Comments state. The document could undergo changes as needed.

2. Introduction

Ubiquitous computing attempts to merge virtual and physical worlds into one seamless environment. It aims to combine an array of software, hardware and physical entities into next generation computing environments. These environments consist of intelligent rooms or spaces, which contain powerful stationary computers, mobile handheld devices and other devices ranging from smart dust to plasma displays. Apart from devices, there may also be a large collection of software services like authentication services, discovery services and location services. There is literally a cornucopia of devices and services that can populate an Active Space. This large collection of devices and services must be coordinated and access to them made simple.

A user in a ubiquitous computing environment should be able to use the numerous entities (ie. devices and services) that exist around him. He should be able to find out easily which entities are available, what their properties are, choose an entity to use and use these entities in a uniform way. This involves a combination of a Discovery Service and a Directory Service. The Discovery Service, as the name suggests, would be responsible for discovering entities, and the Directory Service would collate the information it receives from the Discovery Service and allow meaningful searches on the entities present.

This document describes the Directory Service in the Active Spaces environment. It is called the Space Repository. The Space Repository is present in every Active Space and manages all the data about the space. Briefly, the chief functions of the Space Repository include maintaining a list of all the active entities in the Space, keeping this list up-to-date and providing ways of searching through this data.

We have used XML for describing entities, mainly because it has become a de-facto standard for data representation. It enables easy interoperability of different applications. It also allows easy browsing of the data on the web.

The Space Repository maintains XML descriptions of all the entities currently active in the space. This information is maintained in the form of an XML file that contains a list of all the entities in the space with their descriptions.

3. Requirements

- ***Centralized Repository containing data about an Active Space***

The Space Repository is a centralized database that maintains information about the entities that exist in an Active Space. It is centralized in the sense that there is only one Space Repository per Active Space.

- ***Keeping information up-to-date***

Once the Space is up and running, it is possible for new entities to enter the space and existing entities to leave the space or go down. The Space Repository must reflect the current state of the space. It should make use of the Discovery service for doing this. There must be mechanisms by which the Space Repository can get detailed XML descriptions of entities as and when they come up in a Space.

- ***Searching and Browsing Capabilities***

Since the Space Repository has all the current information about the Active Space, it should be able to handle queries about entities in the space. This search capability can be extended to allow browsing of all the entities in the space.

4. Description

- ***Description of entities***

The Space Repository maintains an XML file containing the list of entities present in the space and their descriptions. These descriptions include the name, location and additional textual information about the entity. The type of the entity (ie. whether it is a device or a service) is also stored. Each entity in an Active Space is a Model, an Incarnation, a Controller or an Adapter depending on the role it performs in the Ubiquitous Application Model. This role of the entity is also stored along with the principal data type that it uses for either input or output. For example, the datatype for an MP3Controller is MP3 and that for a tickertape is ASCII. We also store the name of the host machine on which the entity is running, any resources that it requires, the category of the entity and the driver name for the entity. Resources required by the entity could include serial ports, USB ports, etc. The category refers to what the entity does on a semantic level. The driver name is a reference to the dll or the .so file which has the code for the entity.

- ***Discovery Process***

One of the key requirements of the Space Repository is that it must reflect the current state of the Active Space. The Space Repository uses the Discovery Service to enable it to keep track of which entities are currently active in the Space. The Discovery Service is responsible for tracking software components, people, and physical entities present in an Active Space. The Discovery Service is composed of two sub-services: the Presence Service and Informer Service. The Presence Service keeps track of software entities

currently active in a space, while the Informer Service is responsible for user and physical entity detection. Both services use leases to determine whether or not a particular entity is still active. The Space Repository only makes use of the Informer Service at present – since, it is only concerned with services and devices and not with people at present. However, it would not be difficult to extend the Space Repository to also keep track of people, or to develop a service similar to the Space Repository that keeps track of people.

The Space Repository and Discovery Service communicate through events. The Discovery Service sends events whenever it detects that the state of one or more entities has changed. We felt this helped introduce a degree of decoupling between the two services. Besides, it also enabled other services to listen for discovery events and take any action.

Software components are responsible for sending heartbeats at a pre-established rate to renew their leases. The Presence Service receives these heartbeats and is responsible for generating two new events: start and stop. The start event is sent whenever a new entity is detected in the space, while the stop event is issued when components fail to renew their leases.

The Space Repository listens to the Discovery Channel where in events announcing the starting or stopping of entities are sent by the Discovery Service. It gets a reference to the Discovery Channel from the Event Manager which manages all the channels in a space. The events contain information about the entity - like the name of the entity and its UCR. This information is present in the heartbeats that the entity keeps sending periodically. On getting the UCR of the entity (and from that, its IOR), the Space Repository can query the object for the XML file that describes it. All entities implement the getInfo() method which returns a reference to the XML file describing it. On receiving a "started" event, the Space Repository gets the XML description of the entity and updates its internal database of active entities in the space. Similarly, on receiving a "stopped" event, it removes the entity from its internal database.

- ***Searching the Space Repository***

Since the Space Repository has all the information about the Active Space, it can handle queries about entities in the space. The Space Repository does a search on the XML descriptions of entities it has. The Space Repository can return all the devices in a space or all the services in a space. This enables browsing of entities in the space. For more refined searching, we make use of the CORBA Trading Service (also called a Trader). The Trading Service, which is a standard CORBA service, allows searches where the search conditions are specified by a constraint language. We make use of this Trading Service to allow searches for entities that satisfy various constraints. Any queries that are sent to the Space Repository in the form of constraints are forwarded to the Trader, which processes them and sends the results back to Space Repository, which in turn sends it back to the client.

We have to make sure that the Trader has the same information as the Space Repository. When a new entity is discovered, the Space Repository accesses the description of the entity and populates the trader with its properties. Also, when the Space Repository receives an event indicating that a particular entity is no longer active in the space, it deletes the entry for this entity in the trader.

The use of the Trader is purely for the purposes of implementing a good search engine. A good XML-based relational database could have served the purpose just as well, if not better.

5. Interfaces

Add detailed description of how to use the service and examples

- ***Registration of entities***

Normally entities are registered in the Space Repository through the discovery mechanism. When the entity is started, it automatically starts sending heartbeats which are the Discovery Service keeps listening for. The only thing that is required of the entity is that it implement the getInfo() method returning a reference to an XML file that contains its description.

However, if an entity wants to be registered in the Space Repository outside the Discovery process, it can make use of the externalRegistration() method that the Space Repository implements. This method registers the entity both in the XML database and the trader.

```
void externalRegistration(in Gaia::Entities::Entity entity, in string fileName, in string name, in string ucr)
                                                                    raises (CouldNotRegister);
```

The parameter entity is a reference to the entity object
The parameter fileName is the name of the file having the XML description of the entity
The method can throw an exception CouldNotRegister if it failed to register the entity in the Trader

- ***Querying the Space Repository***

For purposes of querying, there are 3 functions supported by the Space Repository

1. To gets the XML description of an entity, use the function

```
File getDescription(in string identifier);
```

where the parameter _identifier is the name of the entity
It returns the XML description of that entity (if it is present in the Space Repository) as an array of CORBA::Octet

2. To get a list of entities that belong to a specific type (eg. the type could be "Services" or "Devices"), use the function

```
ListOfEntities getListOfEntities(in string type);
```

where the parameter type refers to the type of entities requried

3. For more refined querying, use the function

```
CosTrading::OfferSeq getEntities(in string query);
```

It returns a list of entities that match the query sent by the user. This method queries the trader.
Te parameter query is a string based on the constraints language defined by the trader.
It returns an OfferSeq, which is a type defined by the trader which contains a list of entries that matched the query.

- ***XML Descriptions of entities***

Every entity needs an XML description in order for it to be included in the Space Repository.
An example of the XML description of an entity is

```
<?xml version="1.0" standalone="no"?>

<entity type="Device">
<Name>Badge Detector</Name>
<Location>2447</Location>
<Info>Badge Detector</Info>
<Category>Location</Category>
<Resource>RS232 1</Resource>
<Host>srg192.cs.uiuc.edu</Host>
<DriverName>CORBA/AirIDLtDevice</DriverName>
<MICAType>Model</MICAType>
<DataType>AIrIDLt</DataType>
</entity>
```

The type of the entity refers to whether it is a ”Device” or a “Service” – these are currently the two possible values that the type attribute can take. It can be extended to include “Users” at a later stage if required.
The Name, Location and Info tags give the name, location and some sort of information about the entity.

The Category tag refers to what the entity does on a semantic level.

The Resource tag refers to any hardware resources the entity requires.

The Host tag gives the machine on which the service or the device is running, In the case of a device, it would give the machine to which the device is connected.

The DriverName gives the driver for the service or device. It is a reference to a dll or .so file which has the code for the component.

The MICAType is based on the ubiquitous application model – so the value of this field can be model, incarnation, controller or adapter depending on what role the entity plays.

DataType refers to the type of data (input or output) that is used by the service or device.

6. Conclusion

The Space Repository plays a crucial role in the running of an Active Space. It gives information about the state of an Active Space in terms of what entities are present. It allows applications to discover what exists in a space and make use of the resources in the space. It also makes it easier to construct applications with different models, views, controllers or incarnations to be constructed on the fly.