Space Repository & Active Space Web Browser

Anand Ranganathan

{ranganat@cs.uiuc.edu}

Overview

This document describes the design and implementation of the Space Repository and the Active Space Web Browser. The main function of the Space Repository is to maintain a list of all the entities in the room that are currently active as well as the XML descriptions of these entities. It also provides ways to retrieve this information. The Active Space Web Browser contacts the Space Repository for information about the entities present in the Space and displays it on the Web.

Introduction

There's a wide variety of data in Active Spaces - descriptions of objects, services, users, event logs, security policies, etc. What is needed is a way of managing all this data in a standard way. This is where XML comes in.

XML defines a standard interoperable and extensible framework wherein all data can have structure and be validated. Also, a variety of tools have been developed for parsing and manipulating XML documents. Thus XML is a logical way to go for describing Active Space data.

Every Active Space has a Space Repository that stores XML descriptions of all entities that are present in the space. During bootstrapping, the Space Repository is passed the XML descriptions of the various devices in the Space that have to be started off. It parses these XML files to get the required information for starting the devices and starts them off.

The Space Repository also provides methods for accessing the stored XML data. It is thus possible for clients to get information about the state of an Active Space by querying the Space Repository. Information about the entities in an Active Space can also be accessed through the web. This has been implemented using Java Servlets. Users can thus browse through the different entities present in an Active Space using a web browser.

As of now, XML descriptions have been developed only for devices present in the system – though later, descriptions will also be developed for services, users and other entities.

Functionality of the Space Repository

The Space Repository has two major functions. The first is during the bootstrapping of the Active Space, when the entire Active Space starts up. The second involves the retrieval of information from the Space Repository, more specifically through the Active Space Web Browser.

BootStrapping

In the Bootstrapping of an Active Space, the Space Repository is started off by the Active Space Object (which is responsible for starting off all services in an Active Space). The Space Repository is given two parameters for initialization. One is the Active Space ID, and the other is the name of an XML file that contains a list of entities to be started off on the room. An example of such an XML file is:

```
<room name="2447">
	<device>Air1</device>
	<device>Air2</device>
	<device>EZ1</device>
	<device>EZ2</device>
	<device>Evid</device>
	<device>X10</device>
</room>
```

In the above file, Air1, Air2, EZ1, EZ2, Evid and X10 are names of devices. These devices have XML descriptions that give information about the device as well as information required for starting off the device. For example, the XML description of the device Air1, which is a badge detector is in a file called Air1.xml and goes as follows :

```
<?xml version="1.0" standalone="yes"?>

<entity type="AirIDLt">
	<Name>Badge Detector 1</Name>
	<Location>2447</Location>
	<Info>Badge Detector on Manuel’s desk</Info>
	<Type>Location</Type>
	<Host>srg192.cs.uiuc.edu</Host>
	<Resource>RS232 1</Resource>
	<DriverName>AirIDLtDevice</DriverName>
	<Parameters>-range maximum</Parameters>
</entity>
```

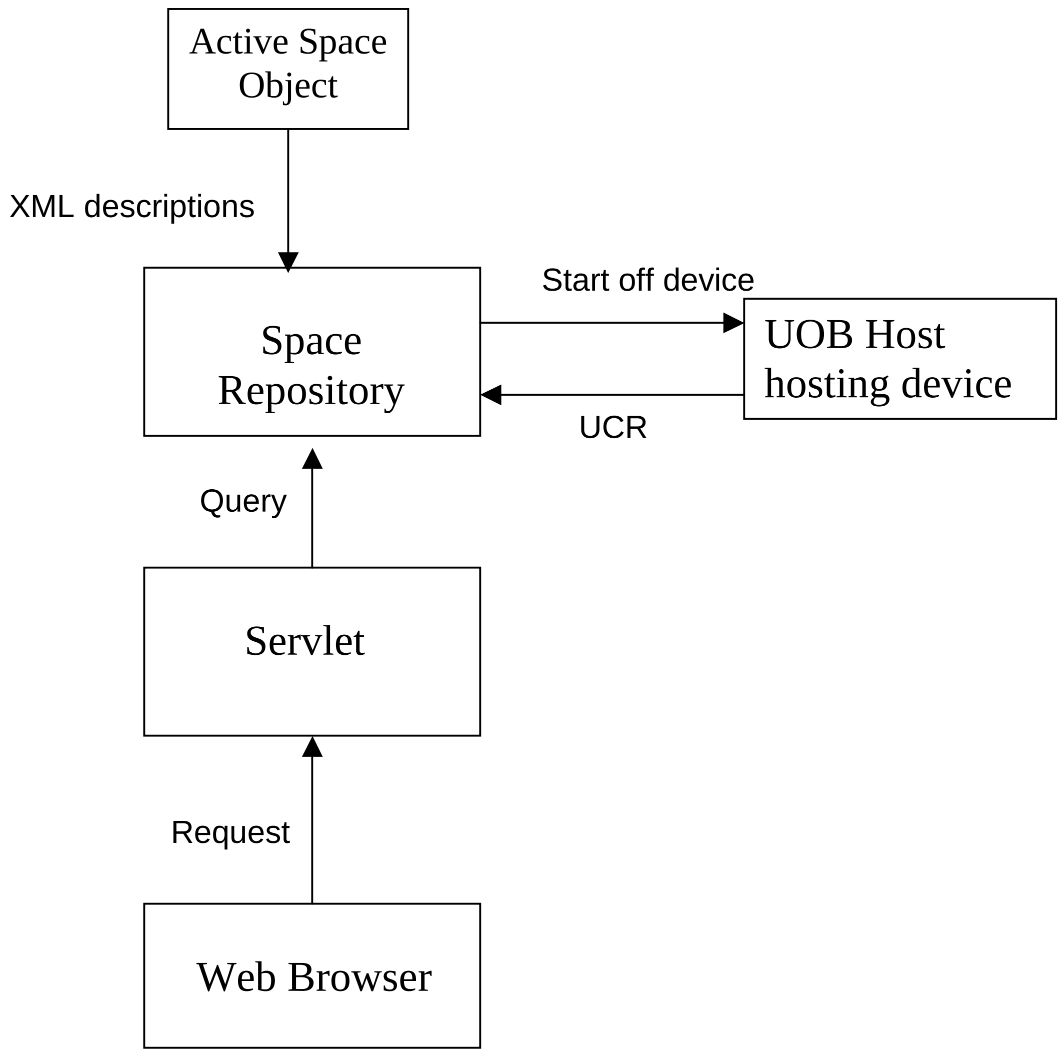
The Space Repository parses the XML documents to get the information required for starting off the devices (ie. the host name, the resource it uses on the host, the driver it needs and any other parameters that are required). It gets the UCR (Unified Component Reference) of the object associated with the device.
The Space Repository also maintains an XML file describing all the entities present in the room. This is done so that even if the Space Repository goes down, all the information is still recoverable from disk - so once the Repository comes back up again, it reads these stored descriptions and gets up-to-date. An example of such a file is :

```
<entity type="room">
        <Name>2447 DCL</Name>
        <Location>2'nd floor</Location>
        <Info>Room with HDTV</Info>
        <Devices>
        <device type="AirIDLt">
                <name>Air1</name>
                <UCID>UCR:srg192.cs.uiuc.edu/1/CORBA/AirIDLtDevice/0</UCID>
        </device>
        <device type="Service">
                <name>PowerPoint</name>
                <UCID>UCR:srg192.cs.uiuc.edu/1/CORBA/PPTService/0</UCID>
        </device>
        </Devices>
</entity>
```

Active Space Web Browser

One of the functions of the Space Repository is to allow retrieval of the information it has. A convenient way of presenting the information in the Space Repository is on the web. Hence, the Active Space Web Browser. This browser allows easy accessing of the current status of an Active Space. It provides some information about the Active Space and lists the different entities present in the Space. It allows the user to get more information about individual entities by just clicking on these entities. Thus a sort of “browsing” of the Active Space is enabled.

The Active Space Web Browser has been implemented using Java Servlets. The servlet captures HTTP requests for descriptions about various entities. It then contacts the Space Repository for the description of that entity. It gets back an XML description of that entity. Using XSL stylesheets, it converts the XML document into a HTML document and sends it back to the web client for displaying on the browser.



The above diagram shows how the different components interact around the Space Repository.

Implementation Details:

The Space Repository has been implemented as a Component in the Unified Object Bus. The Space Repository was built as a CORBA component. The language of implementation was C++.
The descriptions of the various entities and devices was done in XML format. The Apache Xerces XML parser in C++ (XML4C ver 1.3) was used for parsing the XML files.
The source code includes the following files :

1. SpaceRepository.idl
2. SpaceRepository_i.h
3. SpaceRepository_i.cpp
4. PrintDOM.cpp

The compilation for both Solaris and Windows are available in the Build subdirectory.

The Web Interface used Java servlets as its backbone. The Sun Java Servlet Development Kit 2.0 was used for this purpose. The servlets also functioned as CORBA clients to the Space Repository. The idl compiler used to generate stubs for the java clients was jidl (from ORBACUS). It's available at /projects/2k/bin/java/Solaris/jidl.
The source files are :

1. RepositoryServlet.java
2. TAOClient.java
3. TAO_Helper.java

The servlets used XSL (eXtensible Style Language) for easy translation of XML documents to HTML format. XSL defines ways in which XML formats are converted to HTML. This allows for rapid translation of XML to HTML as well as easy modification of the translation strategies. The Sun XSLT Compiler (XSLTC) Version 2 was used for converting the XSL stylesheets into lightweight, portable Java byte code.
The source files are:

1. DefaultRun.java

The java bytecode corresponding to the XSL stylesheets are in

1. room.class
2. devices.class

Future Work:

1. Make the Space Repository listen to the Discovery Channel so that it can know whether any new entities have entered the space or any existing entities have left the space. It can then update itself to reflect the current status of the Active Space.
2. Develop an XML-based search engine for entities in an Active Space – this would make a lot sense when an Active Space has thousands of different entities.
3. Extend the Space Repository to include XML descriptions of all entities in the system – ie. we can have XML descriptions of not only devices but also users, services, places and real world objects.