Connecting Pervasive Frameworks Through Mediation

Florence T. Balagtas and Cedric Angelo M. Festin

Department of Computer Science
University of the Philippines
Diliman, Quezon City, Philippines
florence.balagtas@up.edu.ph, cmfestin@up.edu.ph
http://engg.upd.edu.ph/cs/index.html

Abstract. Context information helps an application decide on what to do in order to adapt to its user's needs. To easily develop ubiquitous applications, there has been increased research in the design and development of frameworks called pervasive computing frameworks. Although these frameworks help application developers create ubiquitous applications easily, interoperability has been a problem because of the different representation of context information and protocols used. This research attempts to solve this problem by creating a Context Information Mediator (CIM) which will serve as a translation gateway between different applications created using different frameworks. To test our system, we developed two versions of an inventory system application that keeps track of items inside a building. The idea here is to let these applications communicate with each other and share information through the CIM.

1 Introduction

Pervasive computing is a computing paradigm that aims to make digital environments composed of ubiquitous applications that are sensitive, responsive and adaptive to human needs without humans actually knowing what happens in the background [1]. Creating ubiquitous applications is quite difficult since different types of devices and different forms of data are to be processed and should be able to work seamlessly. To simplify the creation of ubiquitous applications, several researches in the area of pervasive computing are focused on the creation of pervasive computing frameworks such as the Aura Framework [2], Context-Toolkit framework [5] and One.world framework [6]. These frameworks aim to collect raw data from diverse devices and process the collected data into context information. These context information are then disseminated to diverse applications that run on different devices with the concern for security to avoid unauthorized use of these information [7]. The problem now lies in the representation of context information in different frameworks. Different frameworks have different formatting of context information which prevents them from sharing information. The need for sharing of information among different frameworks is important to provide interoperability which is one of the major goals of pervasive computing.

The problem of interoperability has been present in several areas of computing and different types of mediator systems have been developed in order to address this problem. The *Context Interchange Architecture of Database systems (COIN)* [8] architecture has tried to detect and reconcile semantic conflicts among different database systems. The *P2P gateway* [9] has aimed to facilitate information sharing among different peer-to-peer file sharing systems that uses different protocols. The *Internet architecture* has been designed to facilitate the sharing of computer resources present in different networks through the use of the gateway.

This research aims to create a *Context Information Mediator* (CIM) which is used to get information from different servers that uses different frameworks, and convert these information into data which can be understood by the other frameworks. The CIM framework is developed by using the Java 2 Platform API and uses XML to represent the data in the system. The principles of the Internet architecture were applied when designing the protocols of the CIM.

2 Design and Implementation

This section discusses the steps to achieve the goal of creating a Context Information Mediator.

2.1 Inventory Application Implementation

The application that we have developed is an inventory system for an office area which similar to the Smart Toolbox and Smart Tool Inventory of [11]. What our inventory system application does is that, it keeps track of where a certain item in the inventory is located inside a building. This is done by having sensors monitoring certain areas of the building in order to know where certain items are located. For the purpose of this research, we will simulate the environment that contains the items and the sensors through a graphical user interface (GUI) made using Java Swing. The GUI will show a building that contains three rooms, wherein each room contains several inventory items. Dragging the items in our simulator simulates movement of an item from one location to the next.

The inventory system is composed of two major components. The *pervasive inventory system* which gets the location of the inventory items and stores this information, and the *application interface* which represents the client applications that subscribes to the pervasive inventory system in order to get information about the items. We have created two versions of this sample inventory application. The first version was created using the Aura Contextual Service Interface version 2.3 of the Aura framework. The second version was created using the Context-toolkit framework 2003 release. The idea here is to let the two applications that are written using the two different frameworks communicate with each other and share information through the Context Information Mediator (CIM).

2.2 Context Information Mediator Implementation

The Context Information Mediator architecture consist of two general communicating components, the client and the CIM server. Figure 1 shows a general view of the CIM architecture. Take note that although the figure shows only two clients connected to CIM, the CIM server can handle many clients. We have chosen to design our system base on the client-server model of computer networks since it is already a well established model and is mostly used in networking today. The client-server model is a design in computer networks in which client machines request and receive service by querying the server. The server then sends the needed information to its clients. This model is especially effective when clients and server have their own special tasks that they routinely perform.

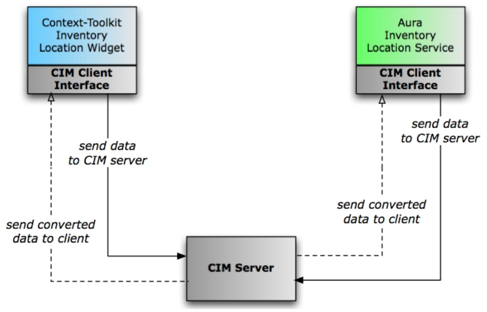

Context-Toolkit Inventory Location Widget
CIM Client Interface
Aura Inventory Location Service
CIM Client Interface
send data to CIM server
send data to CIM server
send converted data to client
send converted data to client
CIM Server

Fig. 1. General view of the CIM architecture

CIM Client Interface. The *CIM Client Interface* sits below the client[1] system and is responsible for forwarding and receiving information to and from the CIM server. In this part of the CIM architecture, we used the *layering strategy* that is used in computer networks.

CIM Server. The *CIM Server* is responsible for gathering all the context information and sends the converted information to its clients. Table 1 shows the subcomponents of the CIM server and their functions.

CIM Data Packets. The CIM data packet contains the information sent between the clients and the CIM server. The CIM packet is divided into two main sections, the *header* and the *data* section. The header section is further subdivided into two which is the *packet type* and the *client type*. The *packet type* identifies what type of packet is sent while the *client type* identifies what type of client has sent the packet. The *data section* contains the main data sent by

[1] In this section, the term *client* refers to the different servers implemented in different frameworks that requests and sends context information to the CIM server.

the sender of the packet. Table 2 shows the different packet types and their descriptions.

Table 3 shows the different types of senders supported by CIM. This helps CIM distinguish from which type of sender the packet was from and helps it decide on how to convert the data contained in the packet.

Table 1. CIM Server Subcomponents

SubComponent Name	Function
Client Request Listener	Responsible for listening to client requests to connect to CIM. It grants the client's request and transfers the request to the Client Registration Manager.
Client Registration Manager	Responsible for asking the client for registration information such as client type (Aura or Context-Toolkit) and is also responsible for assigning a unique identifier to the client. Once the registration process has been done, it assigns a Client Thread Manager to that particular client and the client can now send and receive context information from the CIM server.
CIM Client Thread Managers	Responsible for handling context information received from a particular client assigned to it and stores the data to a central repository. It is also responsible for converting and sending context information that is not present in the client. Aside from that it also monitors if a client is still active by sending AYA (Are You Alive) messages to the client. Once it has detected the client has been disconnected, it informs the main CIM server that which then terminates that client thread manager.

The data section of the CIM packet contains data that is formatted in XML. Although both Aura and Context-toolkit applications convert their data to XML, they still have different representations of the information modeled in XML.

Shown in Figure 2 is a snippet of the XML file in the Aura framework that contains the item *FAX* with item ID number 10004 located at room MH 215 together with the time it was moved.

Shown in Figure 3 is a snippet of the XML file for the Context-Toolkit framework that contains the item *iMac* with item ID number 10001 located at room MH 215 together with the time it was moved.

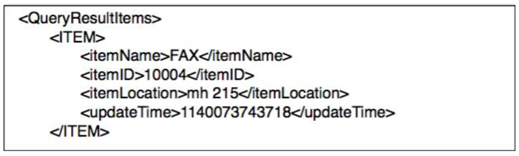
CIM Translation. This section will describe how the data is translated from the client to the CIM server and vice versa.

Table 2. CIM Packet Types

Packet Type	Description
REGREQ	Sent by the CIM Server to request for registration information from the client
REGOK	Sent by the CIM Server to the the client if registration is successful
REGFAILED	Sent by the CIM Server to the client if registration failed
REGDETAILS	Sent by the client as reply to the REGREQ packet. This contains information regarding the client
GOODBYE	Sent by client/server to signify termination
NEWCONTEXTINFO	Sent by client/server that contains the new context information
AYA	Sent by client/server to ask if a client/server is still alive
IAA	Sent by client/server as a reply to the AYA packet

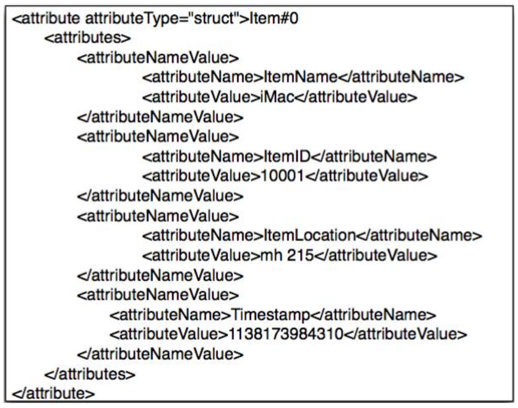
Table 3. CIM Sender Types

Sender Type	Description
AURACLIENT	Sender is from the Aura Framework.
CTKCLIENT	Sender is from the Context-Toolkit Framework.
CIMSERVER	Sender is the CIM Server.



```
<QueryResultItems>
    <ITEM>
        <itemName>FAX</itemName>
        <itemID>10004</itemID>
        <itemLocation>mh 215</itemLocation>
        <updateTime>1140073743718</updateTime>
    </ITEM>
```

Fig. 2. XML format in Aura



```
<attribute attributeType="struct">Item#0
    <attributes>
        <attributeNameValue>
                <attributeName>ItemName</attributeName>
                <attributeValue>iMac</attributeValue>
        </attributeNameValue>
        <attributeNameValue>
                <attributeName>ItemID</attributeName>
                <attributeValue>10001</attributeValue>
        </attributeNameValue>
        <attributeNameValue>
                <attributeName>ItemLocation</attributeName>
                <attributeValue>mh 215</attributeValue>
        </attributeNameValue>
        <attributeNameValue>
            <attributeName>Timestamp</attributeName>
            <attributeValue>1138173984310</attributeValue>
        </attributeNameValue>
    </attributes>
</attribute>
```

Fig. 3. XML format in Context-Toolkit

CIM Client Interface Sends Data to CIM Server. One of the responsibilities of the CIM Client interface is to send new context information obtained by the client system to the CIM server. Before it sends the new data to the CIM server, it first extracts the DataObject (for Context-toolkit clients) or QueryResult object (for Aura clients) that contains the new information from the client system. It then starts to create a CIM packet that will be sent to the CIM server. The CIM packet is created by first appending the necessary headers. The following headers will be added to the packet: NEWCONTEXTINFO for the packet type and AURACLIENT or CTKCLIENT header for the client type. Finally, the XML form of the object extracted from the client is then appended to the data section of the packet. The XML form of the DataObject of a Context-toolkit client is created by using the XMLEncoder class of the Context-toolkit API. It creates an XML form of the DataObject that contains the necessary tags and inventory item information. For the XML form of the QueryResult object of an Aura client, it is created by using the CimXMLEncoder class of the CIM API. The CimXMLEncoder is adapted from the XMLEncoder of the Context-toolkit API. After the packet has been created, it is then sent to the CIM server.

CIM Server Translation. After the CIM Server has received a packet of type NEWCONTEXTINFO from its clients, it first determines the client type of the packet. It then forwards the data section of the packet to the translator that handles the translation of data for that certain client type. After the inventory item information has been extracted from the packet, the inventory database that contains all the inventory data gathered from the different clients are then updated. After the inventory database has been updated, the CIM server will create packets that contains the latest inventory information and will send it to the other clients. In this case, since there are two types of clients currently supported by the CIM framework, the CIM server will create two types of packets, one for the Aura clients and one for the Context-toolkit clients.

CIM Client Interface Receives Data from CIM Server. When the CIM Client receives new context information from the CIM Server, it then extracts the data from the packet and creates it into a DataObject (for Context-toolkit clients) or QueryResult object (for Aura clients). Client applications have a choice of calling several methods from the CimClient class to get the new data. Table 4 shows the methods and their description.

CIM Protocols. The CIM architecture has several protocols for establishing connection between client and server, terminating a connection and the exchange of data between client and server.

Establishing a Connection with the CIM Server. To establish a connection between the CIM Server and the CIM client, the client connects to the CIM Client Request Listener, which then forwards the request to the CIM Client Registration Manager. The CIM Client Registration Manager asks the client for registration information. If the client has successfully sent all the requirements, the

Table 4. CIM Client class data retrieval methods

METHOD	DESCRIPTION
getNewItemListFromCim()	This method returns a java.util.Vector object that contains the list of items in the inventory. This can be used by either an Aura client or a Context-toolkit client.
getNewItemListFromCimForAura()	This method returns a QueryResult object that contains the list of items in the inventory. This is for Aura clients only.
getNewItemListFromCimForCtk()	This method returns a DataObject object that contains the list of items in the inventory. This is for Context-toolkit clients only.

CIM Client Registration Manager sends it a REGOK message that signifies its successful connection to the CIM server. It then assigns a CIM Client Thread manager that is responsible for communicating with the connected client. In cases wherein a client was unable to satisfy the requirements, the CIM Client Registration Manager sends the client a REGFAILED message and disconnects the client.

Terminating a Connection with the CIM Server. In cases wherein the client application leaves or if the CIM Server terminates, each sends a GOODBYE message in order to signal the other that it is leaving. These scenarios show cases wherein there is a clean termination of both client and server. However, there are cases in which either of the two crashes and will not be able to send a GOODBYE message. To resolve this problem, we have created the AYA (Are you alive) packet that is constantly sent by both server and client to each other in order to monitor if the other still exists. The client/server that receives this type of message should reply with a IAA (I am alive) message in order to signify that it is still alive. In cases wherein the client/server is unable to reply with an IAA message, the sender of the AYA packet will terminate its connection with the dead client/server.

Context-Information Exchange. The CIM client sends the CIM server new context-information about its system. It does this by sending a packet of type NEWCONTEXTINFO which contains the new information. When the CIM server receives this type of information, it then updates its database and creates packets containing the newly updated information for both Aura and Context-toolkit clients. It then sends the packets to the other clients connected to CIM.

3 Performance Evaluation and Results

The experiments that we have conducted runs a single CIM server and one or more CIM clients. The first batch of experiments have been done by using an

Apple Powerbook, with a 1.5 Ghz PowerPC G4 processor and 512 MB DDR SDRAM. All applications both client and server have been run in this single computer. For the second batch of experiments, we run the CIM server in an Intel PC, with a 2.66 GHz Pentium 4 processor and 512 MB RAM, while all the clients run on the Apple Powerbook. The clients and server are connected via a local area network (LAN). To start the experiments, we run the CIM server which waits for clients to connect to it. We then run the environment simulator that shows a graphical user interface of a building with three rooms, and the eight inventory items. Each inventory item will be dragged from one room of the building to the next to simulate movement of items. For the experiments that we have done, we have focused on the following parameters that are relevant to the analysis of CIM's performance. These parameters are data size, number of clients connected to the CIM Server and the variety of clients connected to the CIM Server.

Data Sizes. To get the data sizes of the packets, we have written the data that is sent by a client to the server to a file in order to get the number of bytes a single packet contains. The packet size varies based on the type of client that creates the packet (Aura or Context-toolkit), and it also varies based on the number of inventory item information the client has stored in the packet. In order to test how the packet gets bigger as the number of items increases, we have created packets containing 0 items up to 8 items in the inventory. We have done this three times in order to get the average data sizes of the packets sent. Figure 4 shows the relationship of data sizes (in bytes) to the number of items that a packet contains. The number of items here pertains to the number of inventory items that a certain client has information about. The Y-axis describes the average data sizes of packets in bytes. The X-axis shows the number of items contained in a packet. The graph shows the data sizes in bytes for both Context-toolkit and Aura client. Observe that as the number of inventory item information increases, the data size of the packets also increases. Also observe that the Context-toolkit data packet is bigger compared to the Aura data packet. This is because the data of the Context-toolkit client which represents the XML file being sent has more tags. The average data size of a single Context-toolkit packet is 528 bytes per item, while for a single Aura packet, we have an average of 154 bytes per item. There is a slight variation on the data size of each packet since there is no standard size on the information inside a packet.

Translation Time. To get the translation time of the CIM server, we have done a variety of experiments by changing the values of the data sizes sent, changing the number of clients connected and changing the variety of clients connected. To test the effect of data size to the translation time of the CIM server, we iteratively moved items to a particular room monitored by a client one by one every n-milliseconds. As this happens, the client sends its updated inventory information for the room that it monitors to the CIM server. At this point, we measure how long the CIM server can convert these information base on the adding and removing of items. For this experiment, we got the average

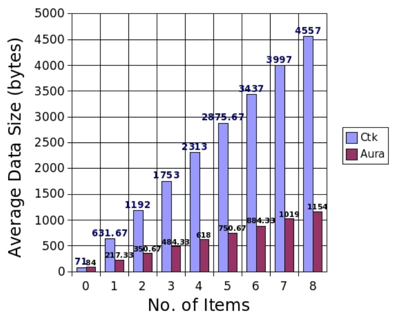


Fig. 4. Average data sizes of packets

translation time by running this test 5 times. We did this for both Aura clients and Context-toolkit clients.

Translation Time vs. Data Sizes. The graph in Figure 5 shows the relationship of the data size versus the time it takes to translate a certain packet for both Context-toolkit and Aura clients. As seen in the graph for both clients, as the data size gets bigger, the translation time also increases. The average translation time per data item for a Context-toolkit client is about 52.59 msecs/item, while for the Aura client we have 52.08 msecs/item. We can see that it takes more time to translate data for a Context-toolkit client as compared to an Aura client, however, the difference is very minimal.

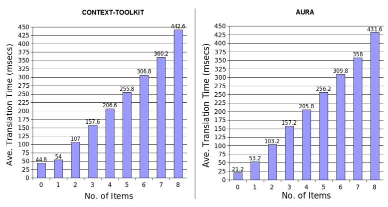


Fig. 5. Average Translation Time vs. Data Sizes

Translation Time vs. Number of Clients connected. To test the effect of number of clients to the translation time of the CIM server, we run several clients that monitors different rooms and continuously sends information about the rooms they monitor. The items are moved from one location to the next every n-milliseconds. To get the total translation time, we add up all the translation times for moving from the first room to the next and vice versa for all clients. We then get the average translation time by doing this experiment 5 times. We increase the number of clients as we did this experiment. We did this for 1, 2, 4, 8, 12 and 16 clients. We did the same experiments to test whether the variety of clients has some effect on how CIM translates data. However, we also varied the types of clients connected to the CIM server and the number of clients per variation. In Figure 6 we can see in the graphs, as the number of clients increases from 1, 2, 4, 8, 12 to 16, the average translation time of the CIM server also increases. The average translation time for one Context-Toolkit client is 211.5 msecs/client, while for the Aura client we have 208.63 msecs/client. The average translation time of the Context-toolkit clients is higher as compared to the Aura clients. This means that it takes longer time to translate the data given by the Context-toolkit clients. This is due to the fact that the data of the Context-toolkit client is more complicated as shown in section 2.2 as compared to the data of the Aura clients.

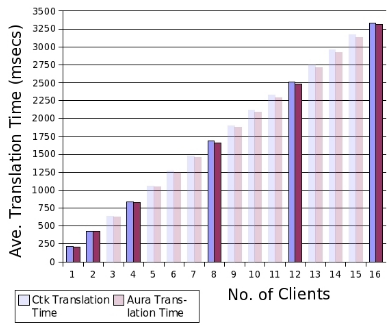


Fig. 6. Comparing Average Translation Time of both Context-toolkit and Aura Clients

Translation Time vs. Variety of Clients. The graph shown in Figure 7 shows the average translation time based on the variation of clients that the CIM server has to deal with. Based on the graph, we observed that for the experiments that has the same types of clients that CIM deals with, the Aura group of clients has a lower translation time as compared to the Context-toolkit group of clients. Comparing the results in which CIM has both a Context-toolkit and an Aura client, the translation time is higher when there are more Context-toolkit clients connected. We can see here that the variation of clients does not really affect the translation time of CIM, however, it is greatly affected by the type of clients that it has to deal with.

Scalability and Extensibility. The scalability of the CIM architecture is measured in terms of the number of clients that the CIM server can accommodate. To test the scalability of CIM, we have tried running 1, 2, 4, 8 and 16 clients connected to CIM and measured CIM's translation time given the number of clients. Please refer to Figure 8 to see the average translation time of clients. As we have seen in the graph, the translation time is doubled as the number of clients double which means that there is an increase in translation time. However, since the increase in translation time is minimal, the performance of the CIM server is not greatly affected. For this experiment, we only tried up to 16 clients, however,

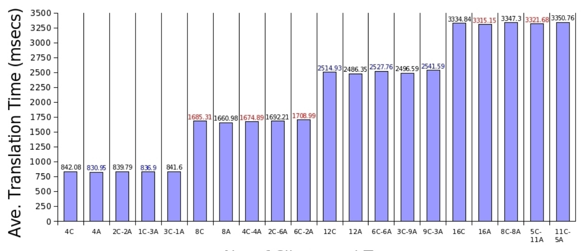


Fig. 7. Average Translation Time vs. Variety of Clients

based on our observation on the increase of translation time, the CIM server can accommodate more, especially if the CIM server is placed in a more powerful computer.

The extensibility of the CIM architecture was measured in terms of how it can be extended to support other frameworks. We have written two different applications using the Context-toolkit framework and the Aura framework which can interoperate using the CIM framework. CIM can support other types of clients that are implemented using the Java platform.

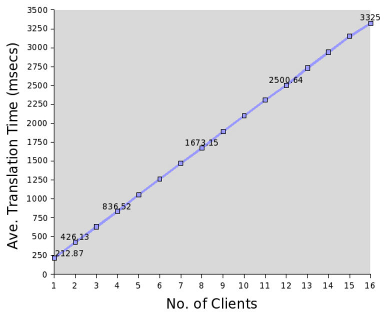


Fig. 8. Average translation time is doubled as the number of clients double

Summary of Experiment Results. Based on the results of our experiments we have the following observations: (1) The data size (in bytes) of packets increases as the number of inventory item information increases. Packets coming from Context-toolkit clients have a bigger data size than that of Aura clients. (2) The translation time increases as the data size of packets increases. (3) The translation time increases as the number of clients connected to the CIM server increases. (4) The translation time of Context-toolkit clients is higher as compared to the Aura clients. This means that it takes more time to translate data coming from a Context-toolkit client as compared to Aura clients. (5) The variety of clients connected to CIM has minimal impact on the translation time of CIM. (6) The CIM framework is scalable in terms of the number of clients it deals with. (7) The CIM framework can be extended to support other types of clients as long as they are using the Java platform.

4 Conclusion

The goal of this research is to create a Context-information Mediator which is used to obtain information from different servers that use different frameworks, and convert these information into data which can be understood by the other frameworks. We have created a Context-information Mediator that serves as a gateway between different frameworks in order for them to share information. Even though CIM was designed to be a gateway for different frameworks, it can still act as a server for clients of the same framework. This can serve as a server for those clients so that they do not need to create their own servers in order to pass information.

The *CIM client interface* serves as the link of a client to the CIM server which abstracts the different protocols needed by the client in order to connect to CIM. The *CIM Server* serves as the translator of different context-information that is provided to the clients connected to it.

We applied the principles of the Internet architecture when designing the protocols of the CIM. We used XML format for the representation of information from the different clients. The choice for choosing XML to represent our data is that it is already a well established standard and is mostly used by different pervasive frameworks such as Context-toolkit, Aura and One.world frameworks.

5 Future Work

For further development of the Context-Information Mediator architecture, the following ideas are suggested:

1. *Translation at End Points and Hybrid Design.* The current implementation for the translation part of the CIM framework is by having the clients send their own format of data to the CIM server and have the CIM server translate and process the data. Another option for the implementation of the translation part of our system is by having the end-points (clients) translate their data into a standard format understood by the CIM server. When data arrives at the CIM server, it does not have to do any translation. The advantage for this design is having the burden of processing distributed among the clients. However, for pervasive computing, we are not assured if the clients have enough processing power since a client system can reside in any type of device. Another option is to have a hybrid design in which a client can choose if it wants to do its own processing or if wants the CIM server to do the processing.
2. *Support for other pervasive computing frameworks.* Currently, the CIM framework supports clients created using the Aura framework and the Context-toolkit framework. CIM can be extended so that it will support other pervasive computing frameworks as well.

3. *A network of CIM servers.* Create several CIM servers that are connected to each other and allow the sharing of information across different CIM servers. This can help in distributing the processing of information for the different clients connected to the different CIM servers. These CIM servers can also act as data filterers wherein a client can choose not to share all of its information to the other clients in connected to the system.

References

1. Saha, D., Mukherjee, A.: Pervasive computing: A paradigm for the 21st century. Pervasive Computing, IEEE **36**(3) (2003) 25–31
2. Sousa, J.P., Garlan, D.: The aura software architecture: an infrastructure for ubiquitous computing. Technical Report CMU-CS-03-183, School of Computer Science, Carnegie Mellon University (2003)
3. Sousa, J.P., Garlan, D.: Aura: An architectural framework for user mobility in ubiquitous computing environments. In: Proceedings of 3rd IEEE/IFIP Conference on Software Architecture, Montreal (2002). (2002)
4. D., G., D.P., S., A., S., P., S.: Project aura: toward distraction-free pervasive computing. Pervasive Computing, IEEE **1** (2002) 22–31
5. Salber, D., Dey, A.K., Abowd, G.D.: The context toolkit: Aiding the development of context-enabled applications. In: Proceedings of CHI'99, ACM Press (1999) 434–441
6. Grimm, R., Davis, J., Lemar, E., Macbeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., Wetherall, D.: System support for pervasive applications. ACM Trans. Comput. Syst. **22**(4) (2004) 421–486
7. Chen, G., Kotz, D.: Solar: A pervasive computing infrastructure for context-aware mobile applications. Technical Report TR2002-421, Dept. of Computer Science, Dartmouth College (2002)
8. Goh, C.H., Bressan, S., Madnick, S., Siegel, M.: Context interchange: new features and formalisms for the intelligent integration of information. ACM Trans. Inf. Syst. **17**(3) (1999) 270–293
9. Lui, S.M., Kwok, S.H.: Interoperability of peer-to-peer file sharing protocols. SIGecom Exch. **3**(3) (2002) 25–33
10. Clark, D.: The design philosophy of the darpa internet protocols. In: SIGCOMM '88: Symposium proceedings on Communications architectures and protocols, New York, NY, USA, ACM Press (1988) 106–114
11. Lampe, M., Strassner, M., Fleisch, E.: A ubiquitous computing environment for aircraft maintenance. In: SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, New York, NY, USA, ACM Press (2004) 1586–1592