A Flexible, Lightweight Middleware Supporting the Development of Distributed Applications across Platforms

Nelson Baloian[2], Gustavo Zurita[1], Pedro Antunez[3] , Felipe Baytelman[1]
Universidad de Chile, Santiago, Chile, [1]Information Systems Department - Business School,
[2] Department of Computer Science.
gnzurita@facea.uchile.cl;nbaloian@dcc.uchile.cl;bayltex@gmail.com
[3]Department of Informatics of the Faculty of Sciences of the University of Lisboa,
Bloco C6, Campo Grande, 1700 Lisboa, Portugal, paa@di.fc.ul.pt

Abstract

Middleware supporting the development of distributed systems has been produced since the beginnings of the Internet. With the emergence of mobile computing new requirements for this kind of middleware arise since the scenario for mobile computing is very different from the desktop computing one. Nowadays, because increasing ubiquitous computing, new scenarios in which desktop computing application should communicate with mobile application are becoming more and more frequent. In this paper we present a platform which apart of being lightweight and easy to use has also the advantage that it enables the communication of software objects between the most used two platforms for mobile devices: JME using the Java language and .Net using C#.

Keywords: Techniques, methods and tools for CSCW in design, Middleware, Mobile.

1. Introduction

Since the early days of distributed, collaborative applications development many authors recognized the need for middleware in order to ease the programming of this kind of software. Sun's RPC (Remote Procedure Call) [11] schema and the CORBA [12] architecture are among the first and most known platforms. Many other followed, each one fulfilling different requirements [1]. They differ with regard to the distribution schemes of the shared data, communication mechanisms, and application architecture they support [2]. Rendezvous [13] and Suite [14] are groupware platforms, which use a central distribution scheme for the data of collaborative applications. MatchMaker [16] uses a replicated distribution scheme. DSC [5] is a p2p groupware system with decentralized topology for supporting synchronous collaborations based on JXTA.

The last years have witnessed an explosion of new collaborative systems for mobile devices that incorporate und utilize their communication capabilities to support collaborative work in ways that were not conceived before or were impossible to implement with desktop computers. Applications allowing users to collaborate in real time over wireless connected mobile devices building ad-hoc networks have attracted the attention of many authors. Some scenarios for which these applications have been developed are:

- Rescue efforts can be more easily coordinated in emergency situations and disaster areas where a wired infrastructure is not available
- People attending a meeting can share ideas and data by means of their mobile devices [18].
- Field survey operations in remote areas with no fixed infrastructure can be easily facilitated.
- A team of construction workers or garden designers on a site without a network infrastructure can share blueprints and schematics [8].
- Educational activities involving students and teachers in collaborative room environments [9].

All these scenarios share common requirements, like high mobility, dynamic user group configuration, easy input procedures, data sharing, etc. However, the use of mobile devices which can provide the needed computing support for the development of these scenarios changed the rules for developing distributes applications since mobile devices have still some problems which are not present on desktop computers. Some of them are described by [10] and [7] and are:

- Low-bandwidth and high latency - Network connectivity of mobile devices depends on radio frequency technologies to exchange data. As a result, wireless networks generally exhibit low bandwidth, latency, and high packet loss rate [3].
- Low processor power - Processor power becomes a limited resource, as mobile devices are designed to be portable and the weight and size of the system must be kept to a minimum.
- Small display size – Most of the mobile devices are equipped with small displays that are not suitable for displaying large amount of information or sophisticated user interface.

Baloian, N., G. Zurita, P. Antunes and F. Baytelman (2007) A Flexible, Lightweight Middleware Supporting the Development of Distributed Applications across Platforms. The 11th International Conference on CSCW in Design, Melbourne, Australia.

- Short battery life - The power necessary for operation, virtually limitless in a stationary device, is a scarce resource for most mobile devices.
- Limited input methods - The possible methods of data input currently available on the market include keyboard, pen-based, and voice. Some devices support varying combination of these methods to give the user the most flexibility [11], although they are still limited compared to their desktop counterparts.
-

This new environment also imposes new restrictions and requirements to the software which runs on them. Novel collaborative paradigms need to be developed to take into account the intermittent application, resource, and user availability, the variability in device capabilities, and the unreliable network connectivity. Accordingly, new middleware supporting the development of distributed communications was necessary and a good number of them have been already developed.

Nowadays, ubiquitous computing is getting more and more prominent and there are many situations which can benefit of the interaction between mobile devices and desktop computers. One example is people engaged in brainstorming-like meeting activities, where they use their handheld devices to input ideas which are collected and displayed by an electronic board [4]. Another interesting example is the use of combined technology (mobile-desktop) in classroom learning activities [3].

Computer technology has made its way into classrooms in a very sound way and it is not uncommon now to see teachers using computers or electronic boards to enhance their lecturing, and students using laptops or other mobile devices to search data, receive and manipulate multimedia-based learning material and work collaboratively [6].

We have been engaged in developing software for supporting in-class synchronous collaborative learning since more than 10 years using a middleware which facilitates the programming of distributed applications called MatchMaker [16]. As mobile devices became an interesting resource to support in-classroom learning we tried to incorporate them by adapting the Java-based systems originally developed for PCs to the new environment. This originated lot of problems. First, it was necessary to develop a lightweight version of MatchMaker for the handhelds, since the original one could not run on JME. Second, freehand writing and sketching input was very uncomfortable because the Java platform was too slow. The same software using pen-based free handwriting and sketching as input programmed in C# was much better to use, producing more accurate writing and sketches. Third, since Java was designed to be platform independent, some of the hardware-dependent features of the handheld were not possible to control from the program.

A first idea to solve this was to use an approach like the one used by XMIDDLE [10], using the object architecture as the only interface between different environments. However this means that every application should implement the conversions between the internal language-dependent object representation and its XML representation. Another solution called SOMU [19] uses web services for exchanging data, but this solution is not lightweight enough and is too slow when implemented across platforms.

The problem of having applications from different worlds talking to each other is certainly becoming a general one, so it is worth to develop a reusable solution. Therefore we opted for developing a new middleware in Java and C# which enables applications living in different worlds share and synchronize data among them in a very simple yet fast enough way. The next section explains the principles used for designing the solution. The following one describes the implementation and the Middleware's API. Since one of our main concerns was the performance of the distributed systems, we made some benchmarking measurements in order to test the suitability of the solution. Then applications developed so far are presented as a proof of concept. Finally we present conclusions and future work concerning the middleware.

2. A trans-platform middleware: principles

As we said, the new hardware and scenarios used by mobile computing imposes other requirements to the communication architecture than those imposed by desktop computing. The main characteristics of the middleware for this new scenario which are different to those developed earlier for desktop applications are:

Decentralized: In many mobile scenarios, the only available network will be the *mobile ad-hoc network* (MANET) provided by the networking capabilities of the mobile devices. This means that the communication and data architecture must follow a peer-to-peer schema avoiding a central server keeping the "master" copy of the data and/or the list of active users. A full centralized schema would be too risky for the mobile scenario because of the communication problems and the dynamic nature of the groups. In [7] and [10] full peer-to-peer middleware for supporting communications in mobile devices are proposed. In [15] a mixed environment is presented, where a non-mobile server can take a different role. A decentralized peer-to-peer schema also adapts itself better to the fact that connectivity between devices is intermittent and the participants list is dynamic, because there is no central server which could leave the session because of a crash or an intermittent communication signal.

Replicated architecture: In distributed software architecture there is no central server keeping a "master" copy of the shared data and the active users list. Therefore, every application must replicate exactly the data others have in order to share a common working environment. This means mechanisms must be implemented in order to synchronize the replicated data.

State-based synchronization: There are mainly two ways to synchronize the data in a replicated environment: by event or by state. Synchronization by event means that all applications start with exactly the same set of data with the same values. During the working session, if one data unit (for example an object) changes its status in one application due to an event caused by the interaction with the user, the application sends this event to all other connected applications, so they can change the state of their object copy accordingly. Synchronizing by state means every time a data unit changes its value, the whole object, and not the event, will be sent to the other applications. If the objects in the application are big, the state-based synchronization mechanism may cause more network traffic than the event-based one. But in an environment where events may not reach all active application or new application instances can join the session at any time, the state-based synchronization is the only reliable choice.

XML-based data exchange: Many of the existing middleware supporting distributed application programming for mobile or desktop devices use standard XML description for an object (like SOAP) in order to transmit it to another application running on another platform. Since different platforms use different internal object representation schema this is the most convenient way for transmitting an object across different platforms. An XML representation of an object may not only contain the names and values of the object variables but also some meta-information describing it, like the class name, which will be used by the other platform to reconstruct the object. Since there are already some "standards" defining the way how an object should be represented by an XML description, we will use one of them in our solution.

3. The architecture of the middleware

The middleware we developed consists of a set of classes implementing an API the programmer can use in order to write distributed applications easily. These classes are available in Java and C# and implement the necessary mechanisms for converting data objects from their internal representations into an XML representation, transmit them across platforms and convert the XML representation into the corresponding internal one. They also provide and start the necessary services for discovering partners in the ad-hoc network and establish connections among the different applications in order to synchronize shared data.

3.1 Discovering partners and establishing connections

In order to have an application join the group of active partners in the ad-hoc network it has to instantiate an object of the *Node* class. The constructor of this node starts a service which will send multicast messages at regular intervals to the group in order to inform other participants of the presence of a new one. It will also start consuming multicast messages from other partners present in the ad-hoc network. This allows the application to maintain a list of active participants updated. Every time a multicast message of a new participant is received, its ID and IP number are stored in the list and a TCP/IP connection is established with that application through which data will be shared. If a certain amount of time has passed without receiving a multicast message from a member of the list of active participants, its entry is deleted and the connection to that application closed. The *Node* class can be extended in order to overwrite some methods. For example, there is *receiveObject* method which is called every time the application receives an object. The figure 1 shows the structure of the communication node implemented by the *Node* class. It has a module which manages threads for sending and receiving multicast packages used to maintain an active partners list. This list is used by another module which is responsible for creating TCP/IP connections with the active partners and destroying them for those partners which left the group and transmit synchronization data.

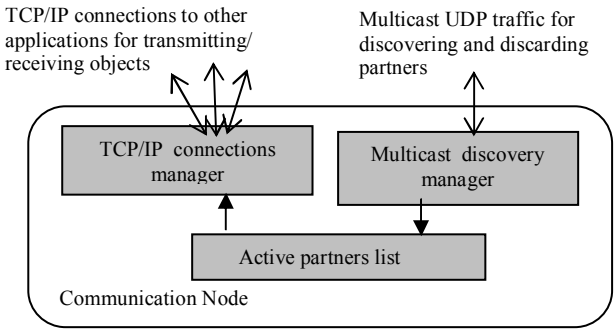

Figure 1: the communication node

3.2 Sharing objects

The data sharing mechanism is based on a "shared objects" principle. A shared object is an abstract class which should be extended in order to create an object class whose state will be transmitted to all active participants when the object changes its state, this is when one or more variables change their value. The programmer implements a shared object by extending the *SharedObject* abstract class. Apart from declaring the field variables and methods for this object, it is often necessary to implement a method called *postProcess* which will be called every time the object state is updated by a remote application. This is a key mechanism which allows the updating of the application's interface when the data changes. Apart from creating a shared object by extending the SharedObject class, programmers have to register it with the communication node giving a name to this object, in order to start receiving the updates from the partners also having a shared object with the same name.

The synchronization of the shared objects is achieved by transmitting a copy of it to all partners every time their state is changed. For this, methods for sending and receiving

objects were designed and implemented. We made these methods public to the API since many small yet powerful applications could be implemented very easily based on those methods without having to use the SharedObject.

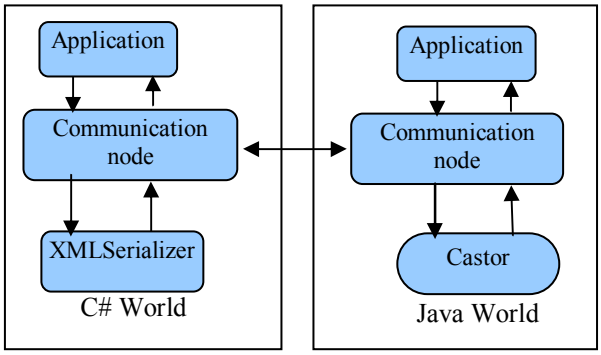

Figure 2: Serializing and transmitting objects

As we said, in order to transmit an object across platforms we need a common language to describe it in a common way. This language will be XML and the representation will be generated in a standard way common to both platforms. In C# this representation can be generated by the *XMLSerializer* library and in Java by the Castor library, both being free and open source software. The fact that the same object should exist in both platform restricts the type of the variables an object can contain to those common to both platforms. In our case there are numeric data, characters, booleans, strings and arrays. Figure 2 shows how the object is transformed into its XML description transmitted and reconstructed between applications running in a "C# world" and another in "Java World". When the application developed by the middleware's user must update the state of an object it is passed to the Node class. This uses the corresponding serializer for producing the XML representation and sends it to the communication node of the other application. The receiving node uses its own de-serializer for transforming the XML representation in the corresponding internal one.

Table 1: Public methods of the middleware's API

For sending/receiving objects	
`public Node(String nodeID, String multicastIP, int multicastPort)`	Creates a Node object which starts the Multicast service for discovering and the TCP/IP server for transferring data.
`public void receiveObject(Object o)`	Used by the communication node in order to receive the objects sent by the partners and synchronize the state of the shared objects.
`public void sendObject(String partnerID, Object obj)`	Sends an object to a certain partner. If partnerID is null the object will be sent to all partners in the network
`public void sendObject(String[] usrIDList, Object obj)`	Sends an object to a list of users
`public void sendToGroup(String groupName, Object o)`	Sends an object to all partners registered in a specific group
Group management	
`public void join(String groupName)`	Joins the application to a certain group characterized by the group's name
`public void leave(String groupName)`	detaches the application from the group specified
`public void remoteJoinGroup(String groupName, String partnerID)`	invokes the join method in a remote application, forcing that application to join a group
`public void remoteLeaveGroup(String groupName, String partnerID)`	invokes the leave method in a remote application, forcing that application to leave a group
Method for shared objects	
`public void postProcess()`	`abstract method of the SharedObjec class. Invoked when the object is updated`
`public void addSObject(SharedObject so, String name)`	`Registers a shard object with the communication node`

3.3 Group management

The learning scenario in a Computer-integrated Classroom was the situation that motivated us to for developing this middleware because of the need to have applications implemented and running in different platforms to share data. In this scenario, we also recognized the need to have the possibility of defining subgroups of partners inside the whole group of active participants. For example, the teacher may want to propose a task which should be accomplished by small groups which do interact among them, but she wants to keep the possibility of looking what the different groups are doing. For this we developed the necessary mechanisms inside the middleware in order to have applications join and leave certain groups. This information is stored in the communication node and is used when the copy of an updated object has to be distributed among participants. Accordingly, we developed the methods which will send objects only to applications belonging to a certain group. An application can join more than one group, so it can receive updates coming from different groups. We also implemented methods for remotely force an application to join and/or leave a group. This was necessary because in many cases, the teacher or the moderator of a group was supposed to form smaller working groups. The teacher or moderator can then join the various in order to "see" what was happening on all of them.

Table 1 shows a description of the most important methods implemented by the middleware. All they are applied to the Communication Node of the application, which is from the Node class or an extended one, except from the last two which are applied to an object of a class extended from the SharedObject class.

4. Benchmarking

Because the performance of the platform was one of our first motivations for the development of the middleware we wanted to test if our solution was fest enough. By fast enough we mean that the time it take an object to be transferred from one application to another does not interrupt the flow of the synchronous work. Of course, this is more or less a subjective evaluation and depends on the application which is being used. An application making intensive use ob the object transfer mechanism may be more sensitive to longer delays than another which sends objects at a slower rate. In any case, it was important to see how long this operation takes in order to analyze which is the permitted delay between the sending of an object and its arrival at the other end our middleware can still support. For this we carried out an experiment in which we measured the round-trip time required to send and object to another application and receive it back. We tested this for objects of different size and between different platforms. We started by sending and receiving objects of 10 bytes up to 10000 bytes first among two applications both running in a C# platform. Then we repeated it for two applications both running in a Java platform and finally we repeated it for two applications. Because the results of these experiments depend on the hardware being used, very standard mobile devices were used in order to have representative results: DELL X50V and DELL X51V. The figure 3 presents the results of these experiments. As we expected, the time required for the round-trip of objects between applications running in similar platforms is much smaller and almost neglectable compared to the time required for the same round-trip between applications running over different platforms. A very interesting and unexpected result was that for all cases, the time is drastically reduced when the objects' size nears the 1000 bytes number. The time remains almost the same for bigger objects. This may be caused by the way the ad-hoc network packs the data in an UDP packages and sends it to the network. In any case, this result gives us a hint on how to design applications in order to have the best response time: objects being shared should contain as much information as possible and in any case they should contain at least 1k bytes.

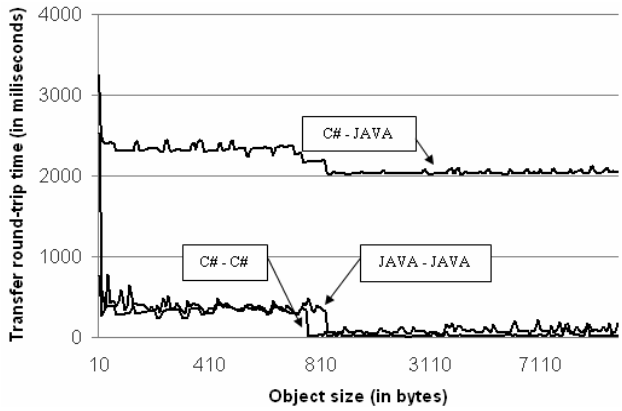

Figure 3: Time required for a round-trip of objects.

5. Implemented applications

With the help of this middleware, we have already implemented some applications for mobile and desktop computers. Some of them make intensive use of the trans-platform feature of the middleware and others were implemented for being run on the same platform. The trans-platform feature was mainly used when we wanted to develop C# applications running on handheld devices communicating with existing applications on desktop computers developed for the Java platform.

MCsketcher: MCSketcher [8] is a system that enables face-to-face collaborative design based on sketches using handheld devices equipped for spontaneous wireless peer-to-peer networking. It is especially targeted for supporting preliminary, in-the-field work, allowing designers to exchange ideas through sketches on empty sheets or over a recently taken photograph of the object being worked on, in a brainstorming-like working style. Pen-based designed human-computer interaction is the key to supporting collaborative work. This application was entirely written in C# aimed for being used only in a mobile situation.

Nomad: Nomad [18] is an Electronic Meeting Support system for handhelds. The design principles applied for developing the system are aimed to help reduce the problems associated with having a small size screen to interact with. The human-handheld interaction is based only in gestures and freehand writing, avoiding the need of widgets and virtual keyboards. The content of the generated documents are organized as concept maps, which gives more flexibility to reorganize and merge the contributions of the meeting attendees. The system is based on handhelds interconnected with an ad-hoc wireless network. This application has a module which allows the use of an electronic board in order to have a common display to show the content being produced during the working session.

Coolmodes: Coolmodes [9] provides a uniform shared workspace environment which allows for constructing and running models with different formal representations (Petri nets, System Dynamics, mathematical graphs etc.) and also supports semi-formal argumentation graphs and hand-written annotations. This software was developed for being used on desktop computers over the Java platform with the goal of being used collaboratively in a classroom. Several students can share a running model by synchronizing their simulation environments. Simulations are analyzed to generate hypotheses about the global behavior of systems. For this system a C# module was developed in order to allow students interact with the software from mobile devices instead of

6. Conclusions

According to the benchmarking results and to the practical experience in using the middleware we can

conclude that this is in fact an easy to use, flexible, and lightweight middleware for developing distributed applications across platforms. Programmers could fast and easily design and program applications. The shared object paradigm was perceived by them as a powerful yet easy to learn and use paradigm.

As we saw from the benchmarking results, the solution is fast enough to implement synchronous applications across platforms for many cases. In all the applications so far implemented with the middleware half a second-delay was not critical for influencing the normal flow of the applications. We are still working in order to make the object transfer time between different platforms smaller, by making the serializing/deserializing process more efficient.

Finally, we want to point out that any platform implementing the API could then also communicate with applications implementing these two platforms using the shared object paradigm.

References

[1] T. Urnes and Nejabi, R: "Tools for implementing groupware: Survey and evaluation", Technical Report No. CS-94-03, York University, 1994.

[2] Lukosch, S: "Adaptive and Transparent data Distribution Support", Proceedings of the CRIWG'02 conference, September, La Serena, Chile, 1002, pp 255-274.

[3] U. Farooq, W. Schafer, M. Rosson, and J. Carroll: M-Education: "Bridging the gap of mobile and desktop computing". WMTE'02, 2002, pp. 91-94.

[4] P. Siland, E. Sutinen. J. Tarhio: "Mobile Collaborative Concept-Mapping Classroom Activity with Simultaneous Field Exploration", Procs WMTE'04, 2004, pp. 114-118.

[5] M. Jianhua, M. Shizuka, L. Jeneung, and H. Runhe: "A P2P groupware system with decentralized topology for supporting synchronous collaborations", International Conf. on Cyberworlds, Singapore, 3-5 December. 2003, pp. 54- 61.

[6] N. Baloian, H. Hoppe, M., Milrad, and M. Hoeksema: "Technologies and educational activities for supporting Challenge-based Learning". WCC 2006, Santiago, Chile.

[7] D. Buzko, W. Lee, W., and A. Helal: "Decentralized ad-hoc Groupware API and framework for mobile Collaboration", Procs of the GROUP'01 Conf. Boulder, USA, 2002, pp. 5-14.

[8] G. Zurita, N. Baloian, F. Baytelman: "A face-to-face system for supporting mobile collaborative design using sketches and pen-based gestures", Proceedings of the CSCWD'06, Nanjing, China, 2006.

[9] N. Pinkwart, U. Hoppe, M. Milrad,: "Educational scenarios for cooperative use of Personal Digital Assistants". J. of Comp. Assisted Learning (2003), 19, 2003, pp. 383-391.

[10] C. Mascolo, L. Capra, S. Zachariadis: "XMIDDLE: A data-sharing middleware for mobile computing. Wireless Personal Communications", 21, Kluwer, Netherlands, 2002, pp. 77-103.

[11] R. Srinivasan: "RPC:Remote Procedure Call Protocol Specification" Version 2. Internet RFC 1831, 2002.

[12] A. Vogel and K. Duddy: "Java Programming with CORBA". John Wiley & Sons (USA) 1998

[13] R. Hill, T. Brinck, S. Rohall, J. Patterson, and W. Wilne, W: "The Rendezvous architecture and language for constructing multiuser applications", ACM Transactions on Computer-Human Interaction, 1(2), 1994, pp. 81-125.

[14] P. Dewan, and R. Choudhary: "A High-level and flexible framework for implementing multi-user interfaces", ACM Transactions on Inf. Systems, 10(4), 1992, pp. 345-380.

[15] R. Litu, and A. Zeitoun: "Infrastructure support for mobile collaboration", HICSS, 2004, Hawaii, USA.

[16] F. Tewissen, A. Lingnau, U. Hoppe, and N. Mannhaupt: "Collaborative Writing in a Computer-integrated Classroom for Early Learning". European Conference on CSCL, Maastricht, The Netherlands, 2001, pp. 593-600..

[17] C. Chang, J. Sheu, and T. Chan: "Concept and design of ad hoc and mobile classrooms", Journal of Assisted Learning, 19, 2003, pp. 336-346.

[18] G. Zurita, N. Baloian, F. Baytelman, M. Morales: "A gestures and freehand writing interaction based Electronic Meeting Support System with handhelds", COOPIS'06 Montpelier, France, November 2006), 2006, pp. 679-696.

[19] A. Neyem, S. Ochoa, J. Pino: "Supporting Mobile Collaboration with Service-Oriented Mobile Units. Procs. of CRIWG'2006, LNCS 4154, pp 228-245.