ME-DiTV: A Middleware Extension for Digital TV

An Architectural Proposal of A Middleware Extension based on Dynamic Context Changes for Distributed System

Victor Hazin da Rocha[1] [2], Felipe Silva Ferraz[1] [2],
Heitor Nascimento de Souza[1], Carlos André Guimarães Ferraz[2]

[1]CESAR – Recife Center for Advanced Studies and Systems
{vhr,fsf,hns}@cesar.org.br

Informatics Center
[2]Federal University of Pernambuco (UFPE) Recife – PE, Brazil
{vhr,fsf3,cagf}@cin.ufpe.br

Abstract—**This paper aims at providing a trustworthy architecture for a middleware extension, based on geolocalized context information, focused on the development of distributed interactive applications for digital TV. The proposed solution was built using the middleware Ginga. Although it has been implemented for the Brazilian Digital TV System, the architecture described in this paper can be applied to other existing Digital TV middleware with the same benefits. Among those, this work presents a project as a study case to demonstrate the solutions viability and performance analyses on its implementation furthermore this works aims to create an easier way to build distributed, context-sensitive applications.**

Keywords-Digital TV;Distributed System; Middleware .

I. Introduction

Most recent data from Brazilian Institute of Statistics shows that 97,2% of Brazilian homes have a Television Device instead of that only 39,3%[1] residences that have a computer. In this scenario, it is possible to realize that the popularity of the television system plays an important role in integration and distributed solutions..

The TV was not originally designed to provide an infrastructure that enables applications and the challenge is increased when we think about distributed applications, whose development is more complex and requires mastery and expertise by the developers [2].

The web pages or applications are usually available 24 hours a day. This is different from the scenario of television programs, which are transmitted only at predefined times by the broadcaster. Therefore, an interactive application sent by the broadcaster is only available to the viewers during the time in which the program is displayed. Thus, depending on the audience of this program and the attractiveness of the application, the application can have millions of simultaneous accesses, overloading broadcaster servers.

This work´s main purpose is to present a middleware extension that can be compatible with different systems and will make development of distributed application easier.

This paper will first present concepts related to Middleware architecture, followed by the proposition of a Middleware Extension. Next, we present and discuss a case study that uses Brazilian Digital Television infrastructure to create a distributed voting system.

The outline of the rest of the paper is organized as follows. Section 1 gives an introduction to the paper. Section 2 describes some middleware concepts. Section 3 describes a few characteristics of Digital TV. Section 4 illustrates the architecture of the proposed solution. Section 5 presents the study case and Finally, Section 6 finishes the paper by explaining a couple of conclusions.

II. Middleware

Distributed systems create new problems that do not exist in centralized systems, like connections problem or network saturation [2]. The question is how to facilitate the development or implementation of distributed applications in such a way that is possible to solve additional problems created by the distribution itself.

In principle, there are different options - from hardware support level to the extension of programming languages to enable support of distributed applications. Software solutions typically provide flexibility because of their suitability for integrating existing technologies (such as operating systems and programming languages). These conditions lead to the concept of Middleware.

Middleware offers general services to support distributed applications execution. The term Middleware suggests that it is software situated between the operating system and the application. Viewing abstractly, Middleware can be envisaged as a “tablecloth” that spreads itself over a heterogeneous network, abstracting the complexity of the underlying technology from the application using it [3].

There are several ways to categorize Middleware. In this paper, we will use the four main types of Middleware found in the literature. These are: transactional, procedural, message-oriented and object-oriented middleware [4].

A. Transactional Middleware

Transactional Middleware supports transactions involving components that run on distributed hosts. This kind of Middleware was designed in order to support distributed synchronous transactions. It should be used when transactions need to be coordinated and synchronized over multiple databases [4].

B. Procedural Middleware

Remote Procedure Calls (RPCs) were designed by Sun Microsystems in the early 1980s as part of the Open Network Computing (ONC) platform. Sun provided remote procedure calls as part of all their operating systems and submitted RPCs as a standard to the X/Open consortium, which adopted it as part of the Distributed Computing Environment (DCE) [5]. RPCs are now available on most Unix implementations and also on Microsoft's Windows operating systems.

According to Pinus [4], RPCs could be used in small, simple applications with primarily point-to-point communication. RPCs are not a good choice to use as the building blocks for enterprise-wide applications where high performance and high reliability are needed.

C. Message-oriented Middeware

Message-oriented middleware (MOM) bear the communication between distributed system components by facilitating message exchange. According to Pinus [4], there are two different types of MOM: message queuing and message passing.

Message queuing is defined as indirect communication model, where communication happens through a queue. A message from one program is sent to a specific queue, identified by name. After the message is stored in this infrastructure, it will be sent to a receiver.

In message passing - a direct communication model - the information is sent to the interested parties. One flavor of message passing is publish-subscribe (pub/sub) middleware model. In pub/sub clients have the ability to subscribe to the interested subjects. After subscribing, the client will receive any message corresponding to a subscribed topic. MOM should be used in the applications where the network or all-components availability is not trustable [4].

D. Object-oriented Middleware

Object-oriented Middleware (OOM), evolved from RPCs, extends them by adding object-oriented concepts. These concepts are: inheritance, object references and exceptions. OOM allows referencing of remote objects and can call operations on them. OOM should be considered for applications where immediate scalability requirements are somewhat limited. These applications should be part of a long-term strategy towards object orientation [4].

III. DIGITAL TV

Digital TV is popular because of the the quality of the image provided by the broadcaster. However, this concept is minimalist. There are three deep concepts of Digital TV: Interactivity, Portability and Connectivity; these concepts, supported by software definitions, are the core of Digital TV [6].

The interactivity and connectivity allows digital TV viewers to submit content and to get a reaction from it. This means it is possible for the viewer to interact with a particular broadcast content [7].

A. Middleware Ginga

Ginga is the name of the middleware specification for the Nipo-Brazilian Digital TV System (SBTVD, from the Portuguese *Sistema Brasileiro de TV Digital*). It consists of a set of standard technologies and innovations which make the most advanced middleware specification and the best solution for the brazilians requirements [6].

The middleware is divided into two main integrated subsystems, which allow the development of applications following two different programming paradigms. Those subsystems are called Ginga-NCL (for declarative NCL applications) and Ginga-J (for imperative Java applications). The use of any of these two paradigms depends on the requirements of each application [6].

In addition to making it possible to send applications to compatible TVs, Ginga provides information about content transmitted to the receiver through a set of tables, called SI (*Service Information*). Among the tables that compose this group we highlight the EIT (*Event Information Table*) and NIT (*Network Information Table*). The EIT is responsible for delivering information related to the program schedule, while the NIT contains information about the network that the content is being made[8].

Ginga-J was chosen to be used in this article because of the support to the network layer of the Ginga middleware.

1) Ginga-J

Ginga-J is designed to provide an infrastructure for the implementation of applications based on Java language, with features aimed specifically for the digital TV environment [9].

Ginga-J, as the name suggests, supports Java procedural language. According to [10] *" it is the logical subsystem of the Ginga middleware responsible for processing imperative applications written using the Java language*".

IV. ARCHITECTURE

This section aims at describing the architecture of the solution proposed by this work.

The first important project decision was the choice for building a message-oriented middleware. This choice was made because MOM systems can provide distributed communication on the basis of asynchronous interaction model allowing the system to continue processing once a message has been sent [11].

Figure 1 shows the macro architecture for the implemented solution. The middleware was divided into three separate layers, which will be detailed below.

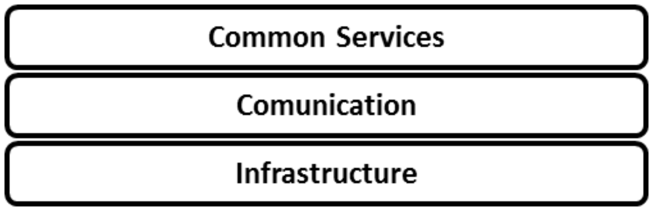

Figure 1. Macro Architecture

The described architecture is compatible with existing middleware such as MHP [12] and Ginga [8]. The goal is to facilitate the creation of distributed interactive applications for Digital TV which can have millions of simultaneous accesses, causing an application to work as a distributed system, dividing the access to the servers based on the context of the device responsible for TVs connection, called as set-top box.

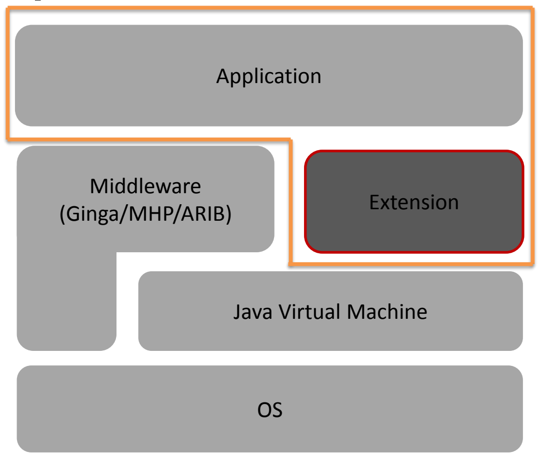

Figure 2. Potential Middleware extension.

To make possible for the extension to be used by existing middleware, the module presented in this paper was developed in Java. This choice was made because Java is the language used on the main Digital TV middlewares, such as MHP, ARIB (Association of Radio Industries and Businesses) and Ginga-J. Despite the fact that the extension is not be incorporated into any TV middleware, it can still have access to Middleware Features since it is presented in the same level as other applications.. Figure 2 shows how the extension is positioned connecting the Middleware and the application, considering as basis a mixed architecture of the MHP/ARIB/Ginga-J. The module presented in this paper should include together with the application as is highlighted in the Figure 2, this is required for not be necessary to change the existing middleware implementations.

For a possible adoption of this extension we choose to use Ginga middleware. Figure 3 shows the usage scenario of the proposed extension as part of a bigger structure.

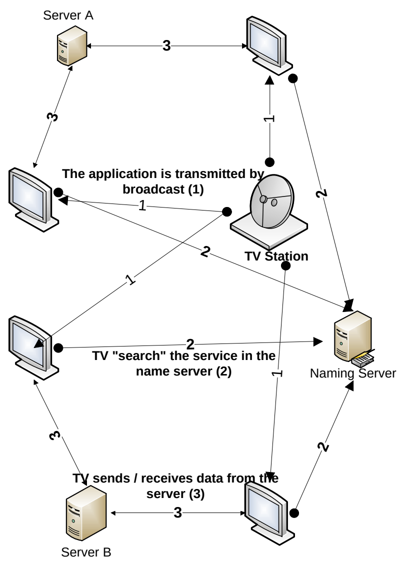
Server A
3
3
1
2
The application is transmitted by broadcast (1)
1
2
TV Station
1
2
TV "search" the service in the name server (2)
Naming Server
1
3
2
TV sends / receives data from the server (3)
3
Server B

Figure 3. Usage scenario

The first step is to build an application that makes use of the feature of distribution based on location provided by the extension; this application must be registered in the naming service and sent by the broadcaster to viewers via broadcast (1). When the application is received by the TV middleware, a query is made to the naming service (2) to discover what is the most appropriate server based on the location to perform information exchanges. When the application receives the reply of the naming service, it can finally exchange information via messages with the broadcaster server (3).

A. *Infrastructure*

Figure 4 shows the class diagram of this layer.

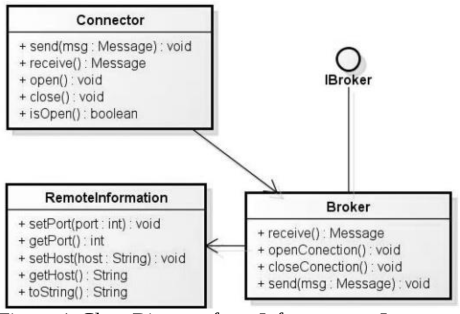

Figure 4. Class Diagram from Infrastructure Layer

This layer is responsible for sending messages to the network layer within the operating system, making transparent communication between processes and applications that uses the middleware and hide the use of sockets from the layers above.

Classes and methods of communication layer cannot be called directly by the developer. It is only used by others layers to send messages over the network.

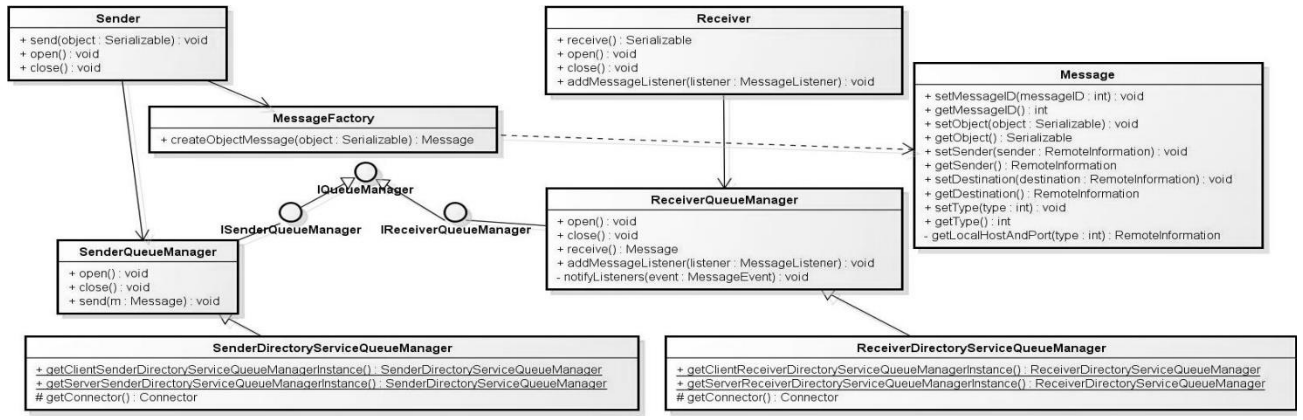
Sender
+ send(object : Serializable) : void
+ open() : void
+ close() : void
Receiver
+ receive() : Serializable
+ open() : void
+ close() : void
+ addMessageListener(listener : MessageListener) : void
Message
+ setMessageID(messageID : int) : void
+ getMessageID() : int
+ setObject(object : Serializable) : void
+ getObject() : Serializable
+ setSender(sender : RemoteInformation) : void
+ getSender() : RemoteInformation
+ setDestination(destination : RemoteInformation) : void
+ getDestination() : RemoteInformation
+ setType(type : int) : void
+ getType() : int
- getLocalHostAndPort(type : int) : RemoteInformation
MessageFactory
+ createObjectMessage(object : Serializable) : Message
IQueueManager
ISenderQueueManager
IReceiverQueueManager
ReceiverQueueManager
+ open() : void
+ close() : void
+ receive() : Message
+ addMessageListener(listener : MessageListener) : void
- notifyListeners(event : MessageEvent) : void
SenderQueueManager
+ open() : void
+ close() : void
+ send(m : Message) : void
SenderDirectoryServiceQueueManager
+ getClientSenderDirectoryServiceQueueManagerInstance() : SenderDirectoryServiceQueueManager
+ getServerSenderDirectoryServiceQueueManagerInstance() : SenderDirectoryServiceQueueManager
# getConnector() : Connector
ReceiverDirectoryServiceQueueManager
+ getClientReceiverDirectoryServiceQueueManagerInstance() : ReceiverDirectoryServiceQueueManager
+ getServerReceiverDirectoryServiceQueueManagerInstance() : ReceiverDirectoryServiceQueueManager
# getConnector() : Connector

Figure 5. Class Diagram from Communication Layer

B. *Communication*

The communication layer is the responsible for the creation of each message that will be send through the network and for the creation and management of queues.

This layer also makes abstract the sending of messages to the application. Figure 5 has the diagram that represents this layer.

Only the Sender and Receiver classes can be called by the application developer, providing the mechanisms of transparency of communication.

C. *Common Services*

This layer is responsible for providing naming service and the location transparency. Furthermore, others services could be provided such as, security service.

The services of this layer are available for use by both the middleware and the application. The class diagram that best describes the architecture can be seen in Figure 6.

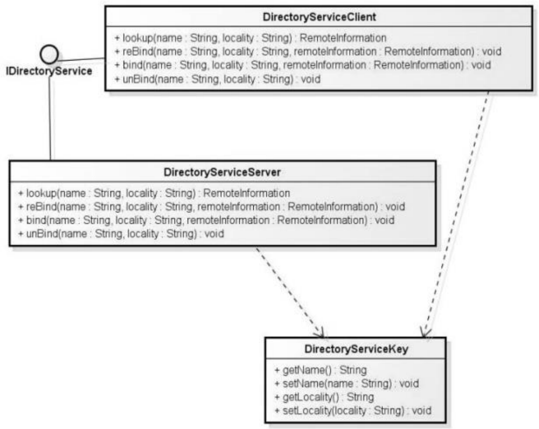
DirectoryServiceClient
+ lookup(name : String, locality : String) : RemoteInformation
+ reBind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ bind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ unBind(name : String, locality : String) : void
IDirectoryService
DirectoryServiceServer
+ lookup(name : String, locality : String) : RemoteInformation
+ reBind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ bind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ unBind(name : String, locality : String) : void
DirectoryServiceKey
+ getName() : String
+ setName(name : String) : void
+ getLocality() : String
+ setLocality(locality : String) : void

Figure 6. Class Diagram from Common Services Layer

The service name of the proposed extension provides four methods for application developers: *bind*, *reBind unBind*, *lookUp*. The bind/rebind/unbind methods are responsible to register/deregister a server/service in the naming service, together with that, lookup method is responsible for naming service and find server/service address using the service name and the information about who is transmitting the application, obtained from the network. Since the information contained in the network is part of the context of the set-top box (or the device), we assume the naming service provided by the solution is context-sensitive

V. STUDY CASE

To validate the architecture proposed in this paper, we implemented a version for an extension compatible with the Ginga middleware, using the Java 1.3. This is the Java version compatible with Ginga [13]. Ginga was chosen because it is the middleware of the Brazilian Digital TV.

The current implementation includes all the features that were described in the architecture section, it contains a total of 24 classes. This implementation contains more classes that were explained in Figure 4, Figure 5 and Figure 6, because some helper classes were created.

A. *Voting System*

To better evaluate the architecture two identical applications were developed. The difference between them is restricted to how they send objects across the network. The App01 is the application that uses the middleware extension built in this work, while App02 does not use the extension.

The application chosen to be developed was a voting system for reality shows. The system receives a vote given by the user / viewer through some input device, in the present case we use remote control and since the input is received, the Middleware takes care of sending to the broadcast server.

As the focus of this work is to facilitate the development of applications making transparent the communication layer, the GUI was not implemented.

The application works by pressing one of the input keys in the remote control. The information of which button was pressed is detected and then sent to the server via message.

```
String vote = "participant01";
Sender.getInstance().open();
Sender.getInstance().send(vote);
Sender.getInstance().close();
```

Figure 7. App01 Code

In Figure 7, we can see the code of App01, which is responsible for sending the vote of set-top-box/television to the server, this application uses the module constructed in this work.

Figure 8 shows the code of App02 that is responsible for doing the submission of the votes. Note that in App02 the server address should be passed with the application, so their location cannot be changed dynamically. In App01, only the parameters defined in this paper must be sent together with the application, and the server location could change dynamically.

```
Socket clientSocket;
clientSocket = new Socket(Constants.APPLICATION_HOST,
Constants.APPLICATION_PORT);

ObjectOutputStream outToServer;
outToServer = new ObjectOutputStream(
                clientSocket.getOutputStream());

String vote = "participant01";
outToServer.writeObject(vote);
outToServer.flush();
clientSocket.close();
```

Figure 8. App02 Code

Looking at the code responsible for the communication of the two applications, one can observe that the code of App01 is much simpler and transparent than the App02. Figure 9 shows the server code responsible for receiving the votes from the App01, while Figure 10 contains the server code of App02.

```
RemoteInformation ri = new
RemoteInformation(Constants.APPLICATION_PORT,
Constants.APPLICATION_HOST);

DirectoryServiceClient.getInstance().reBind(Constants.S
ERVICE_NAME,Constants.LOCALITY, ri);

MainReceiver r = new MainReceiver();
r.init();
public MainReceiver() {
    receiver = new Receiver();
}
public void init() throws InterruptedException {
receiver.open();
receiver.addMessageListener(new MessageListener() {
    public void onMessageReceived(MessageEvent event)
     {
          countVotes(receiver.receive());
     }
  });
}
```

Figure 9. App01 Server Code

Differently from clients, the server of App01 has more code lines than the App02 server. This happens because the middleware extension proposed in this paper enables transparent error handling, and offers a names service, allowing the server to changes its IP address dynamically.

```
int port = Constants.APPLICATION_PORT;
welcomeSocket = new ServerSocket(port);
Socket connectionSocket = welcomeSocket.accept();
ObjectInputStream input = new ObjectInputStream(
connectionSocket.getInputStream());
countVotes(input.readObject());
connectionSocket.close();
welcomeSocket.close();
```

Figure 10. App02 Server Code

In the next subsection, we will present an experiment to evaluate the performance of the applications built here to validate the proposed extension.

B. *Validation and Results*

To analyze the middleware impact, tests were executed to measure the performance and reliability of the two applications. The tests were made in a laboratory of Digital TV with a television embedded with Ginga and a Playout EITV, which is a complete TV broadcast station that can perform transmissions containing TV programs in high definition and interactive content [14]. The configuration of the testing environment is illustrated in Figure 11.

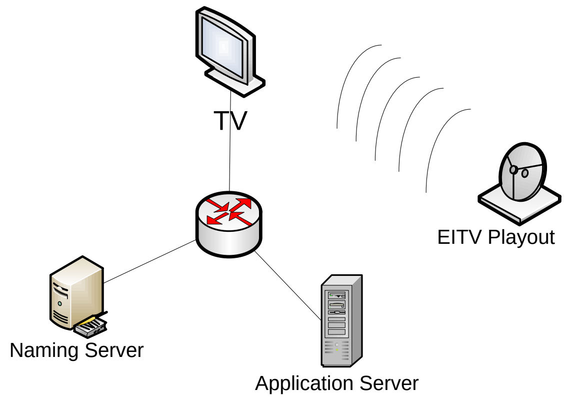

Figure 11. Test Environment

The test was done as follows: for each application, we added a function to send 100 votes consecutively when one of the colored keys on the remote control was pressed.

The modified applications were transmitted one at a time, to the TV using the EITV Playout. For each application, the time between the arrival of the first and last on the server was measured. Each vote was sent separately, a new connection was opened to send the vote, after send it, the connection was closed. The experiment was done to simulate an environment where a user wants to vote more than once. The experiment was repeated five times only due to the stability of the local network of the test environment, and the results are shown in Table I.

TABLE I – TIME IN SECONDS BETWEEN THE ARRIVAL OF THE FIRST AND LAST VOTE ON THE SERVERS

Id	**Time(s) to complete send action on App01**	**Time(s) to complete send action on App02**
1º	32,57	31,17
2º	31,99	31,48
3º	32,49	31,58
4º	32,26	31,73
5º	32,19	31,68
Avg Function	32,30	31,53

Analyzing the results presented in Table I, we can realize that App01 had a delay of less them 2,4% comparing to App02. App01 shows its potential even losing in performance, by using the localization and communication transparency provided by this work proposal.

VI. CONCLUSION

This work proposed an extension that provides a context-sensitive feature to applications, in a way to make easier and more abstract communication for adopted implementations. After a brief explanation about the architecture, a proof of concept was developed and validated using performance tests.

Even though the study case presented was developed for Brazilian Digital Television Middleware, the proposed solution can be adopted in different Middleware, or as a solo API.

Through the analysis of the results, it can be seen that the extension can decrease the performance in less than 3%, but shows its power by creating an easier way to build distributed, context-sensitive applications. Furthermore, it guarantees a more dynamically and network-error free environment since it abstract those scenarios.

ACKNOWLEDGMENT

This work is supported by the CAPES (process 23038-023577/2008-23, AUX-PE-RH-TVD 385/2008).

REFERENCES

[1] IBGE, *Síntese de indicadores sociais : uma análise das condiçoes de vida da populaçao brasileira*. IBGE, 2010.

[2] A. S. Tanenbaum and M. V. Steen, *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002,.

[3] A. Puder, K. Römer, and F. Pilhofer, *Distributed Systems Architecture: A Middleware Approach*. Morgan Kaufmann, 2005.

[4] H. Pinus, “Middleware: Past and present a comparison” , 2004. [Online]. Available: http://www.research.umbc.edu/~dgorin1/451/middleware/middleware.pdf. [Accessed: 20-Sep-2012].

[5] The Open Group, “DCE 1.1: Remote Procedure Call.” [Online]. Available: http://www.opengroup.org/public/pubs/catalog/c706.htm. [Accessed: 20-Sep-2012].

[6] “Ginga Digital TV Middleware Specification,” 2012. [Online]. Available: http://www.ginga.org. [Accessed: 10-Sep-2012].

[7] L. Cosentino, “Software: a essência da TV digital,” in *TV Digital Qualidade e Interatividade*, Brasília: IEL/NC, 2007, pp. 41–49.

[8] “ABNT NBR 15603-1:2007. Televisão digital terrestre - Multiplexação e serviços de informação (SI) - Parte 1: Serviços de informação do sistema de radiodifusão.” .

[9] “Site Oficial da TV Digital Brasileira,” 2012. [Online]. Available: http://dtv.org.br. [Accessed: 10-Sep-2012].

[10] L. F. Soares, “Ambiente para desenvolvimento de aplicações declarativas para a TV digital brasileira,” in *TV Digital Qualidade e Interatividade*, Brasília: IEL/NC, 2007, pp. 51–62.

[11] E. Curry, Message-Oriented Middleware, in Middleware for Communications (ed. Q. H. Mahmoud), John Wiley & Sons, Ltd, Chichester, 2004.

[12] “MHP,” 2012. [Online]. Available: http://www.mhp.org/. [Accessed: 05-Sep-2012].

[13] “ABNT NBR 15606-6: Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 6: Java DTV 1.3.” 2010.

[14] “EITV Playout - Estação completa de TV digital interativa para os padrões SBTVD, ISDB e DVB,” 2012. [Online]. Available: http://www.eitv.com.br/playout.php. [Accessed: 15-Sep-2012].