Using Acme to Specify the Software Architecture of the OpenH323 Open Source Code

FERNANDO NEY C. NASCIMENTO, ANTÔNIO C. THEÓPHILO C. JÚNIOR,
VIRGÍNIA C. C. DE PAULA, GUIDO LEMOS DE S. FILHO
DIMAp – CCET - UFRN
Campus Universitário – Lagoa Nova – 59072-970, Natal/RN
BRASIL

Abstract: This paper presents an Acme specification of the software architecture of the OpenH323 open source code. OpenH323 is a collaborative initiative that aims to produce a public implementation of the ITU-T recommendation H.323. Providing this architectural description, we intended to make easy the maintenance and reuse of this open source code. It is well known that the implementation of a videoconference system requires the conception of a very complex code. One of the approaches to implement such type of system is to reuse code. Considering this, we highlight, as the main contributions of our work, to facilitate and to accelerate the implementation of videoconference systems compatible with the H.323 recommendation through the reuse of the OpenH323 code.

Key-Words: - OpenH323, Acme , H.323, model, Software Architecture

1 Introduction

Nowadays, great achievements have been reached in the scope of telecommunications and computer networks. At the same time, the demand for information has been growing and users look for a better support to use multiple services. The integration of audio, video and text media characterizes the systems called multimedia. Among these applications is the videoconference, which provides a bidirectional exchange, in real time, of audio and video signals between groups of users located in two or more distinct places [8,12,13].

The necessity to produce compatible implementations, forced software manufactures, universities and telecommunications companies to develop standards for this application. In 1996 the ITU-T (International Telecommunications Union) approved a standard for audio, video and data communication in IP networks, published through the Recommendation H.323 [5], allowing the developing of various prototypes and systems in conformance with the standard.

Among these, is the OpenH323 Project [9], which is a collaborative initiative for development an open source implementation of H.323. The OpenH323 Project is important because commercial H.323 implementations are very expensive. Another good point of this project is that its source code is available for development and modification.

Nowadays, if a programmer wants to develop systems using the OpenH323 or wants to join the effort of development of the source code, he must study directly the source code, which is extremely complex. Besides de source code, there are the recommendations and a few documents in natural language, which can contain ambiguities. The direct problem coming from this fact is the developers´ difficulty to understand the source code. As there are not documents describing the software architecture, this fact takes us to propose a model for the architecture of this software.

The existing gap between design and implementation can be fulfilled with Software Architecture, which enables the developers not only to specify requisites but also to structure the applications in a modular way. Our work consists in, using an ADL (Architecture Description Language), to associate elements defined in the recommendation with components of the source code of the OpenH323 Project. Our goal with this project is to make the relationship between the recommendation and the source code developed more accurate, simplifying the understanding of the OpenH323 open source code. This way, facilitating its reuse.

This paper is structured in the following way: section 2 and 3 present the required background for the understanding of our proposal. In section 2 we introduce Acme, the Architecture Description Language that we have used. In section 3 we describe, informally, the ITU-T H.323 Recommendation. In the sequence, section 4 presents our proposal of formalization and specification of the Recommendation H.323 using Acme and its mapping for de OpenH323 source code. After that, we present some conclusions and suggestions for future studies.

2 Software Architecture and Acme

The software architecture of a system defines its high-level structure like a collection of interactive components [2].

The purpose of software architecture is to provide the link between design and implementation. Usually, it is described through an "***A**rchitecture **D**escription **L**anguage* – ADL", which is defined by formals notations used to represent and analyze the properties and architecture of a system in a high level of abstraction [2,4].

The language used in this paper is Acme, a language provided by the Acme Project, which started in 1995. The purpose of the Acme language is not only to provide a formal notation for software architecture description, but also to be an interchange language between different ADLs [1].

There are seven basic constructs in this language. They are: components, connectors, systems, ports, roles, representations and properties. The main constructs are components and connectors. *Components* represent basic computational systems (clients, servers, filter etc.). *Connectors* are the elements that link components. Connectors have *roles* that are attached to the *ports* of components (see Figure 1). The complete description of each construct is beyond the scope of this paper and further information can be found in [1]. Besides the structural definitions, Acme also proposes a graphical notation for its elements. An example of this notation is shown in Figure 1.

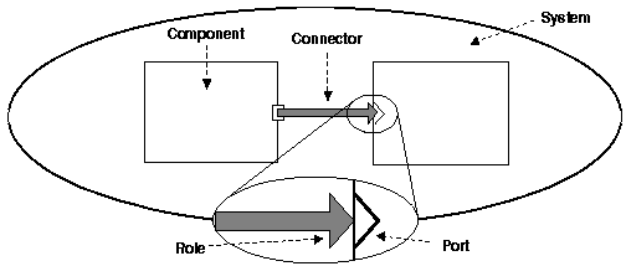

Figure 1 - Main elements of an Acme description.

There is a tool, named *AcmeStudio* [3], available for Windows® environment, that provides a graphical interface to describe a software architecture using the Acme language. Some figures contained in this paper were generated using this tool.

3 ITU-T Recommendation H.323

The videoconference market has been defined by a series of standards published by the ITU-T, known as H.32X. Each standard has a major role in defining the videoconference for different transmission quality levels [13]. Of all the standards produced, the one that deserves more attention is the H.323. It has originally been designed for IP/Ethernet networks but it can also be applied for ATM networks.

The H.323 standard defines four components as its major modules, named Terminal, Gateway, Gatekeeper and MCU (Multipoint Control Unit).

Terminal: is, as its name suggests, an endpoint that provides real-time bi-directional communication with another Terminal, Gateway or MCU.

Gateway: is an endpoint that provides real-time bi-directional communication between Terminals H.323 and other types of ITU Terminals (H.310, H.320, H.321, H.322, H.324, V.70) or another Gateway.

Gatekeeper: this component provides address translation and network access control for Terminals, Gateways and Gatekeepers, besides providing others optional services.

Multipoint Control Unit (MCU): is an endpoint that provides a multipoint conference between three or more Terminals or Gateways. A MCU consists of two parts: a Multipoint Controller and a Multipoint Processor (optional).

Multipoint Controller: is a component that provides a multipoint conference between three or more endpoints (Terminals or Gateways).

Multipoint Processor: is an optional component that provides central processing of audio, video and/or data in a multipoint conference.

Figure 2 show the diagram of a H.323 Terminal. Considering the Terminal as a whole module, it was divided in various interdependent sub-modules:

System Control: Responsible for the videoconference control. This module is sub-divided into three others (RAS Control, Call Control and H.245 Control):

RAS Control: Responsible for the RAS (Register, Admission and Status) functions between Terminals and Gatekeepers. This sub-module uses messages defined in the H.225 standard. It is not used in environments where a Gatekeeper is not present.

Call Control: This sub-module is responsible for the call initialization between two endpoints H.323. It corresponds to the H.225 standard.

H245 Control: Responsible for the exchange of messages between two endpoints. It corresponds to the H.245 standard.

Audio Codec: Responsible for the audio codec. It can be subdivided in sub-modules (G.711, G.722 etc.) where each one is responsible for a specific codec.

Video Codec: Responsible for the video codec. It also can be subdivided in sub-modules (H.261, H.263 etc.) where each one is responsible for a specific codec.

Data Module: Responsible for the handling of data that is neither audio nor video. It is specified in the ITU T.120 standard.

All these sub-modules can be further divided based in the need for detailing.

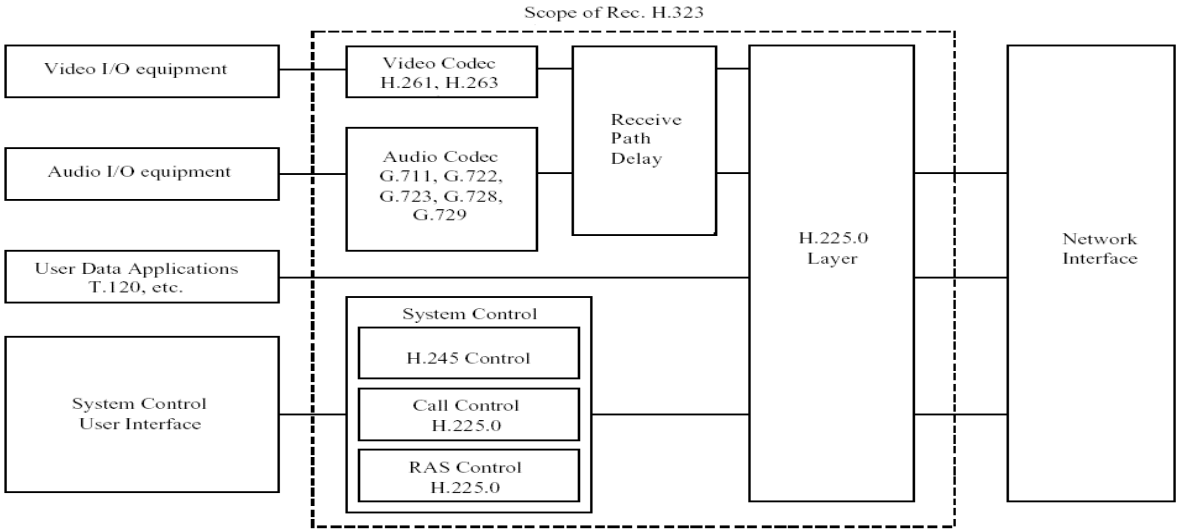

Figure 2. H.323 Terminal diagram.

There are several initiatives dealing with H.323 implementation, some of them under the open source philosophy. Among these, we can highlight the *OpenH323 Project* [9], which has the aim of offering a public implementation of this protocol to be used by researchers and commercial developers. A big obstacle faced by the OpenH323's developers is the lack of a standard documentation that shows what has been done and what is to be done. Even the H.323 standard is very textual and of hard understanding.

These facts motivated us to develop a generic model of a Videoconference using the Acme language, aiming to specify the relationship between the H.323 standard and the OpenH323 source code. Another possible contribution of this work is a documentation of the OpenH323 source code, which would help new and existing developers to produce new code reusing the work already done.

4 Acme Architectural Description of the OpenH323 Project

The first stage in an architectural description consists of a definition of a set of *Acme* special types to represent problem domain elements. The type definition allows modifying system characteristics so that these changes are propagated to the entire model. The use of the AcmeStudio imposes a top-down approach in the architecture specification.

To facilitate design of the videoconference system model, in agreement with the ITU-T recommendation, we defined the following *Acme Families* in our proposal: *H323*, *H225*, *H245*, *Audio* and *Video*. The idea was to declare sets of related types, of distinct sub-systems, that could be useful to other systems. Table 2 shows main *Acme Components* for each *Acme Family*.

Each *component* can and be better specified through R*epresentations* depending on the detail level required [1].

Family	Main Components
H323	SystemControl, AudioCodec, VideoCodec, Data, ReceivePathDelay
H225	RASControl, CallControl
H245	H245Control
Audio	G711, G722, G723, G728, G729
Video	H261, H263, MPEG2

Table 2. Acme Families and its main components.

The global components *Terminal, Gateway, Gatekeeper* and *MCU*, were also defined in order to represent the four main entities of the H.323 recommendation [5]. The configuration of a multipoint conference is depicted in Figure 3, which shows the relationship between H.323 components.

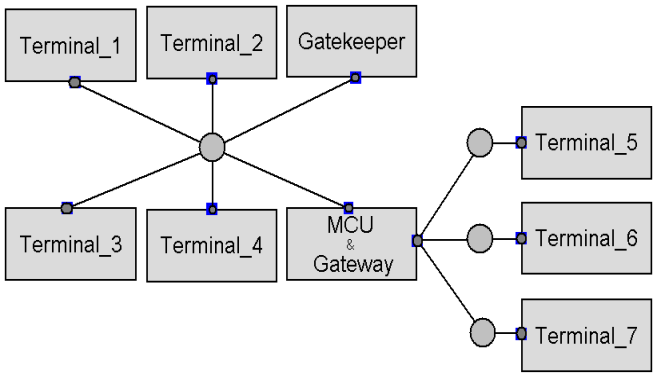

Figure 3. Videoconference configuration.

The communication among entities is represented through the use of *Connectors*. Each *Connector* is associated to a set of *Roles*. One *Role* defines a

participant of the communication represented by the *Connector*. In the *Components*, the roles are attached to a set of *Ports* that identifies an interface between the *Component* and its environment [1].

In this paper we will only to describe in detail the specification of the *Terminal*, due to lack of space. The H.323 Terminal was chosen because its internal components are part of the majority of the other H.323 entities.

In figure 4 we show the *Terminal Component Representation*. Considering that all *Terminals* shall support voice communications, being video and data optional, we create a *Property* to components *AudioCodec*, *VideoCodec,* and *Data* called *Def-Prop* (Default *Property*) of *Boolean* type. This property is set for true for *AudioCodec* and false for *VideoCodec* and *Data*. There is a lower-level (detailed) representation for each internal component of a *Terminal*.

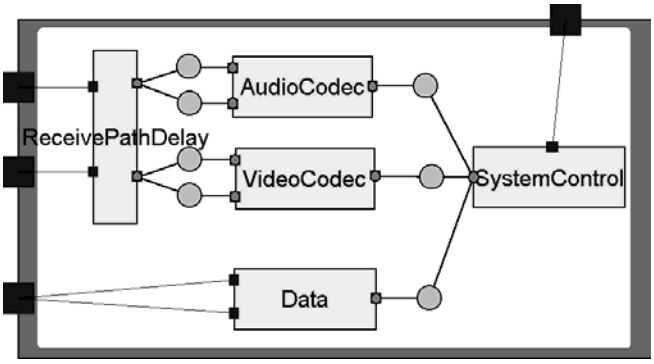

Figure 4. Terminal Representation.

Figure 5 shows the *SystemControl Representation*. In this figure we can see *Bindings,* which are big squares in the *Component* bounds. *Bindings* establish the link between the representation levels of the system. The main *SystemControl* functions exchange *RAS* messages, establish a connection and open or close logical channels between two H.323 endpoints. This representation shows instances of components *RASControl*, *CallControl* and *H245Control*, defined on the recommendations [5, 6, 7]. Figure 6 presents the *Acme* code for the *SystemControl* representation.

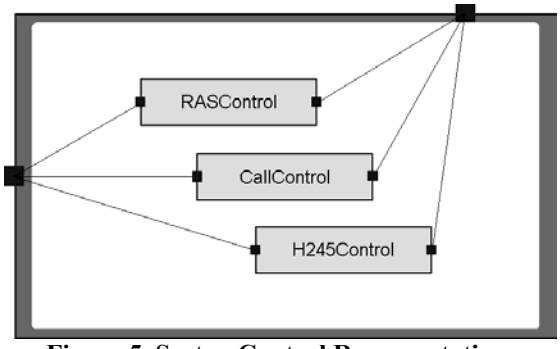

Figure 5. SystemControl Representation.

```
Family H323 = {
 ...
 Component Type SystemControl {
  Port ControlOut : SystemControl_PortOut;
  Port ControlIn : SystemControl_PortIn;
  ...
  Representation {
     System SystemControl_Rep : H225, H245 = {
       ...
       Component RASControl1 : RASControl;
       Component CallControl1 : CallControl;
       Component H245Control1 : H245Control;
       ...
     };
    };
  };
 };
 ...
};
```

Figure 6. Acme H.323 Representation.

Figure 7 shows the *AudioCodec Representation*. Its function is encoding and decoding the audio stream. For each codec there is a specific component in our model. In this representation we see the following audio codecs: G.711, G.722, G.723, G.728 and G.729.

All H.323 Terminals must code and decode speech according to recommendation G.711, other audio codecs may be supported optionally. So, the definition of these types is not enough to specify the dynamism of this architecture. Dynamic elements can be defined through the use of the *optional* keyword. Thus, to specify optional elements, we make its communication ports optional, as shows the Acme code listed in Figure 8.

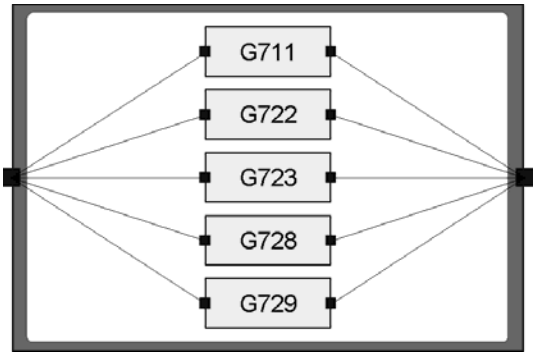

Figure 7. AudioCodec Component Representation.

```
Component G722 = {
 ...
 optional port ActivePortSend;
 optional port ActivePortReceive;
 ...
};
```

Figure 8. Acme code for optional elements.

The *VideoCodec Component* is similar to the *AudioCodec*, therefore, we assumed that is not necessary to describe this component.

Figure 9 shows the *CallControl Representation* that, as stated earlier, has the function to control the signaling between two endpoints.

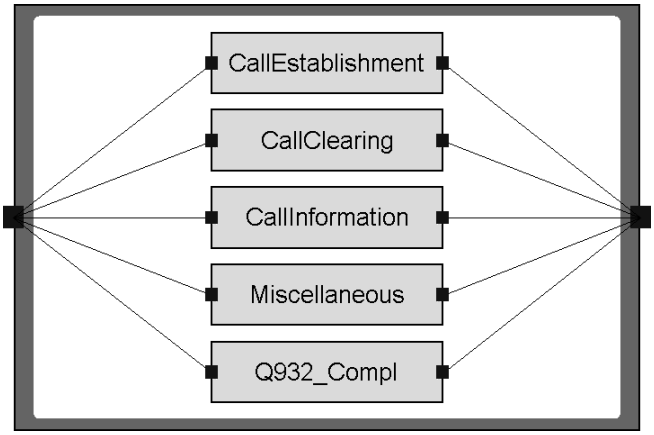

Figure 9. CallControl Component Representation.

From this point we begin to present the mapping between the *Acme* specification and the source code of produced by the OpenH323 Project. We start by the *Terminal Component*.

The *Terminal Component* is implemented in OpenH323 by the ***H323Endpoint*** class. This class has several methods for *Terminal* operation and configuration. The class prototype is shown in Figure 10.

```
class H323EndPoint : public PObject
{
 ...
 public:
  ...
  void AddCapability(H323Capability * capability);
  ...
  BOOL DiscoverGatekeeper(H323Transport * transport);
  ...
  H323Connection * MakeCall(const Pstring &remotePart,
                            PString & token, void *userData);
  ...
};
```

Figure 10. *H323Endpoint* class prototype.

Among the ***H323Endpoint*** methods the more important are:

AddCapability → add a codec to the *Terminal* capabilities table.

DiscoverGatekeeper → discover and select a Gatekeeper.

MakeCall → Make a call to a remote party.

In the *Terminal* representation there is a *SystemControl* component (see Figure 4). The *SystemControl,* after establishing a communication channel (Call Signaling Channel) between two endpoints (invoking the **MakeCall()** method), is used to start signaling.

SystemControl is represented by three components (see Figure 5). *RASControl* uses messages to perform registration, admissions, bandwidth changes, status, and disengage procedures between endpoints and Gatekeepers. This component is only used in network environments that contain a Gatekeeper.

H323RasPDU class implements the RASControl component. he prototype of this class is shown by Figure 11.

```
class H323RasPDU : public H225_RasMessage
{
 ...
 public:
  virtual H225_RegistrationRequest &
          BuildRegistrationRequest (unsigned seqNum);
  virtual H225_RegistrationConfirm &
          BuildRegistrationConfirm(unsigned seqNum);
  virtual H225_RegistrationReject &
          BuildRegistrationReject(unsigned seqNum,
                                  unsigned reason);
  virtual H225_AdmissionRequest &
          BuildAdmissionReques(unsigned seqNum);
  ...
  virtual H225_BandwidthRequest &
          Build BandwidthReques(unsigned seqNum);
  ...
  virtual H225_InfoRequestResponse &
          BuildInfoRequestResponse(unsigned seqNum);
  ...
  virtual H225_DisengageRequest &
          BuildDisengageReques(unsigned seqNum);
  ...
  virtual H225_UnknownMessageResponse&
       BuildUnknownMessageResponse(unsigned seqNum);
  ...
};
```

Figure 11. *H323RasPDU* class prototype.

After RAS control messages have been exchanged, call control procedures take place through call control messages defined in *CallControl* component. H323SignalPDU class implements the CallControl component. Figure 12 shows its prototype.

```
Class H323SignalPDU :
              public H225_H323_UserInformation {
  ...
  H225_Setup_UUIE & BuildSetup(
                const H323Connection & connection,
                const H323TransportAddress & destAddr);
  H225_CallProceeding_UUIE & BuildCallProceeding(
                const H323Connection & connection);
  H225_Connect_UUIE & BuildConnect(
                const H323Connection & connection);
  H225_Alerting_UUIE & BuildAlerting(
                const H323Connection & connection);
  H225_ReleaseComplete_UUIE & BuildReleaseComplete(
                const H323Connection & connection);
  ...
};
```

Figure 12. *H323SignalPDU* class prototype.

Once both sides have exchanged call setup messages, they are able to exchange messages for initial communication and capability exchange defined in the *H245Control* component. The *H245Control*PDU class implements the H245Control component. In this class there is one method for each procedure of the defined by the Recommendation H245. These methods are **H245_XXX** where **XXX** is the name of the procedure.

5 Conclusion

The implementation of a videoconference system is not an easy task. The information available to start the development of this very complex code is the textual description of the H.323 standard. Another option is to reuse open source code produced by projects like OpenH323. But, usually these projects have very poor documentation and are of hard understanding.

Our aim in specifying a H.323 software architecture and associate this description to the source code of OpenH323 Project is to make available a model which offers a better functional understanding of the recommendation and its mapping to the OpenH323´s source code, showing the relationship between its components. Our main contribution is to facilitate future implementations of these kinds of systems.

The utilization of an ADL like Acme assured that the specification was produced considering a standard notation with well-defined semantics. This approach leads to a precise and unambiguous specification of the software architecture. Based on this specification it is possible to make tests of the system before implementing it, taking into consideration aspects like performance and costs.

In conclusion, we believe to have reached the goal of describing a complex and real system under the view of an ADL, simplifying the task of new developers to join the OpenH323 Project effort. Another contribution of this work is to provide to developers who want to start from the H.323 standard a software architecture in a standard language designed for this.

As future study we can incorporate the Acme specification (more detailed) to the set of documents that describes the OpenH323 Project, being available to keep it up to date.

6 References

[1] Garlan, David; Monroe, Robert & Wile, David. **The Acme Architectural Description Language. Copyright**©1998. School of Computer Science. Carnegie Mellon University - 1998. <http://www.cs.cmu.edu/~acme>

[2] Shaw, Mary & Garlan, David. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall. 1996.

[3] Kompanek, Andrew. **AcmeStudio - User's Manual Copyright**©1998. School of Computer Science. Carnegie Mellon University - 1998. <http://www.cs.cmu.edu/~acme>

[4] Medvidovic, Nenodand Taylor & Richard N. **A classification and Comparison Framework for Software Architecture Description Language.** In: IEEE Transactions on Software Engineering, vol. 26, no. 1, p. 70-93, January 2000.

[5] ITU-T. **Recommendation H.323** (11/2000) - Packet-based multimedia communications **systems**

[6] ITU-T. **Recommendation H.225.0** (11/2000), Call signalling protocols and media stream packetization for packet-based multimedia communication systems

[7] ITU-T. **Recommendation H.245** (07/2001) - Control Protocol for multimedia communication.

[8] ITU-T **Recommendation F.730** – Videoconference Service General. ITU-T, 1992.

[9] **OpenH323 Project**, Open Source implementation of the ITU-H.323 teleconferencing protocol that can be used by personal developers and commercial users without charge. 1998 - <http://www.openh323.org>

[10] Tavares, T. A; Leite, J. C, & Souza Filho, G. L. (2000) **Design da Interface do Sistema de Vídeo Sob Demanda da Rede NatalNet**. In: VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia - SBMídia, 2000, Natal. Anais do VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia. 2000. p.141-157.

[11] Queiroz, Carlos Alexandre; Tavares, Tatiana Aires; Souza Filho, Guido Lemos de & Paula, Virgínia Carneiro de. **Arquitetura e Implementação do Sistema Internet-DTV**. In: XXVI Conferência Latino-Americana de Informática, 2000, Cidade do México. Anais do XXVI Conferência Latino-Americana de Informática. 2000.

[12] Oliveira, Jauvane Cavalcante de. **TVS: Um Sistema de Videoconferência** - Dissertação de Mestrado 1996. – Departamento de Informática da PUC/RJ.

[13] Zanin, Fabio A. **Um Modelo para Videoconferência em Computador Pessoal sobre Redes IP** – Dissertação de Mestrado 2000 – Universidade Federal do Rio Grande do Sul – UFRGS – Instituto de Informática.