THE GENERIC CONTEXT SHARING PROTOCOL GCSP

Application to signaling in a cross-network and multi-provider environment

Rony Chahine[1] and Claude Rigault[2]
1 Département informatique et réseaux,
ENST, 46 rue Barrault, 75 013 Paris, France,
Corebridge, 3 rue Saint Philippe du Roule, 75 008 Paris
rony.chahine@enst.fr
2 Département informatique et réseaux,
ENST, 46 rue Barrault, 75 013 Paris, France
GET-Télécom Paris ; LTCI-UMR 5141 CNRS
claude.rigault@enst.fr

Abstract. This paper proposes a new signaling paradigm and a new signaling protocol called the Generic Context Sharing Protocol (GCSP) for the construction of a global control plane over present and future communication networks. After identifying the special nature of the control plane software involved in the setup of a conversational service instance it examines the various mechanisms for information sharing which leads to our new proposal. We show that this new data-based protocol is better suited to control plane requirements than the present day's command-oriented signaling mechanisms. We indicate the basic principles of the protocol and we give a brief description of the generic context. We show the place of this proposal in the present day research efforts and we mention a practical implementation case.

Keywords- Control plane, Signaling, Association, Generic Context, Data based communication, Cooperating Computing, Signaling Protocol.

1 Introduction: From the special nature of control plane software to new signaling mechanisms

In present and future communication networks, a control plane is required when services use a conversational communication paradigm. By “conversational communication paradigm” we mean that a communication environment is established before users start to exchange media and remains established till an explicit release is issued. This communication environment is persistent, and thus state-full, during the whole communication session. Services based on the conversational paradigm require dedicated functions to set-up modify and release the communication environment. We call these dedicated functions “Control Functions” [1], and we define the “control plane” as the set of all processes that execute control functions. The emblematic service using the conversational paradigm is the Plain Old Telephony Service (POTS) where resources are reserved in all participating switches and are freed when one of the participants hangs-up.

In the part 2 of this paper, we analyze the special nature of the software that executes network and service control activities. We characterize this special nature as “Cooperative Computing”, and we examine the requirements of this type of computing. The requirement of interest for this paper is “Information Sharing” by means of signaling.

In the part 3 of this paper, we consider the different approaches to the problem of signaling between partner entities in the cooperative computing situation of the control plane and we come to the conclusion that the current "command based" mechanism is not the most efficient approach to the signaling problem and we show that in a cooperative situation a "data based" approach is more efficient. We therefore propose a "data based" mechanism for signaling leading to a paradigm shift in signaling methods. This proposal is in line with current control plane research efforts. It complements the NSIS [2] IETF work as a contribution to the NSIS Signaling Layer in the signaling protocols for conversational services and its relation to other works will be detailed in the paper. If we come to use a "data based" mechanism for signaling, it becomes necessary to standardize the Call Instance Data or the content of the dynamic session memories in each partner that we call “Local Contexts”.

In the part 4 of this paper, we propose a generic data structure, called the Generic Context, shared by control entities in the same manner as management entities share information in Management Information Bases (MIB) [3] and we give an overview of this generic data structure.

In the part 5 we describe the basic principles of a new signaling protocol called Generic Context Sharing Protocol GCSP, based on the "data based" mechanism for signaling.

In the part 6 we show the place that this signaling protocol could occupy in the scope of present day research in the control plane and signaling area.

Finally in the part 7 we give an implementation example in GCSP in the case of a Computer Telephony Integration application

2 The special nature of control plane software

Control plane software is a very complicated and costly task. The origin of this problem may indeed be traced to the cooperative nature of control plane software. To understand this point we should underline that computer science may be divided into 3 main branches: centralized computing, distributed computing and cooperative computing. In ***Centralized Computing***, a mainframe masters all the processes in a company. All peripherals are intelligence-less slaves executing orders from a single Master computer. In ***Distributed Computing*** many smaller computers, work together, specializing on given tasks and providing some activity independence. This new computing organization requires communication between the computers. The general solution developed by computer science for distributed computing is the "Client-Server" architecture, based on the "request and answer" communication paradigm. However, the client server architecture does not depart fundamentally from the former centralized. The client is mostly concerned by customization and interface problems and the essential service data and service logic are located in the central server position.

A radically new solution to the distribution of intelligence on many computers would be a new kind of computer science called ***"Cooperative computing".*** In this scheme there is no central position, all the computers are equal and no one is in a *permanent* position to give orders to the others. While many different efforts are taking place towards the development of a theoretical solution for cooperative computing, (grid computing, peer to peer processing, agents…), no generally accepted theory has been yet proposed. However some examples of working cooperative applications have been successfully developed. The main one, for our concern, is the "call control" application of telephone switches. Indeed control functions work in a cooperative manner. In the telephone network all switches are equal, there is no centralized platform controlling the setup of a call or its release. Each switch works on a peer-to-peer basis to achieve a global service. It is this special cooperative nature of control activities, and of the lack of a theoretical base for this new type of computing, which leads “Call control applications” to be developed as very complicated ad hoc solutions.

However we may identify some key subjects for research in cooperative computing that are fundamental to control plane software:

- *Cooperative computing requires information sharing between partners*. In the control plane, this Information sharing is called "signaling". It derives that Signaling research is not merely a research problem for telephony; it is a fundamental cooperative computing research problem.

- *Cooperative computing requires the setup of Associations between the partners:* each process involved in a service session has pointers to his partner processes. All pointers, put together, form an association tree that gives a global view of the service session.

- *Cooperative computing requires policies for the distribution of decision authorities.* As there is no central control point, all entities are equal. The difficulty is to decide which entity should take a decision at a given time.

- *Cooperative computing requires behavior models for the partners.* A partner should be aware of the effects of his actions in the other partners. Each partner should have a behavior model of the partners with whom he cooperates. In telephony such behavior models are referred to as "call models".

- *Trust and security.* Authentication and ciphering are required in order to have a safe communication between two cooperating partners.

This list of research problems is certainly not exhaustive, but is sufficient to understand the complexity of control plane activities. The rest of this paper concentrates on the signaling problem.

3 Global Control plane requirement and candidate signaling mechanisms

3.1 The requirement for a global control plane

A service may be designed as a composition of various service components hosted by different service providers. For example in a bank call center, Bob has a user interface on his PC which allows him to search customers profile and to call them directly from his PC. The *Profile-lookup* component and the *call* component are provided by two different service providers and are integrated together to build a richer service with a single user interface. We call this a "multi-provider service". Let's assume further that when a customer calls Bob on his fixed office phone, Bob receives a screen popup on his PC with the customer profile. If Bob is away from his office, he may want to have the calling customer profile displayed on his PDA and take the call from his mobile phone. The service that was available in the bank private network is now extended across several networks. We call it a "cross-network" service. Today, signaling paths are missing both for multi-provider and cross-network services. Partial solutions do exist: web services or other types of middleware achieve some multi-provider services, but they do not apply to heterogeneous networks. Cross-network services require signaling gateways to do the translation from a signaling protocol to another. Cross-network services are considered in [4-7] for a limited set of services and in a more general, but very centralized manner by [8]. The requirements of multi-provider and cross-network services are very difficult to satisfy with existing control plane concepts. We therefore propose enhanced mechanisms that would achieve cross-network compatibility and extend a same "global" control plane over different networks and different component providers to achieve an easier service implementation.

3.2 Candidate signaling methods

During a service instance, control processes store their Call Instance Data (CID) in a temporary memory page that we call a *local context.* This memory page is released when the service instance is terminated. All the partners of a same service instance share the information in their local contexts by means of signaling. The various mechanisms for sharing information have been classified [9] into three different categories: “data based mechanism”, “command based mechanism” and “object oriented mechanism”.

In the ***command based*** mechanism local context data are private and therefore modified indirectly by an incoming command. A control process does not have the rights to read or modify directly a remote context; it uses instead a predefined set of commands. When a process receives a command it performs the necessary actions and modifies its local context.

On the contrary, in the ***data based*** mechanism, also called variable oriented approach, cooperating processes in a same service session can read each other CIDs or local context and thus modify them directly. To make this possible, local context should have a specific data structure that all processes can understand. A solution is to organize local context attributes following a tree structure, like a SNMP MIB, or an object oriented structure like the OSI MIB [10]. We will later show that it is preferable in a cooperative computing environment to use simple commands like Get and Set to read or modify instance data of a remote process instead of using a wide set of commands. Of course security is a requirement of such a mechanism: local contexts can be read only by trusted remote processes. The rights of a process to read and modify instance data in remote contexts should be set according to trust and security policies.

Finally in the ***object-oriented*** mechanism, processes communicate by invoking remote objects located on various machines using the client/server architecture. Examples of this mechanism are web services [11], CORBA [12] and RMI [13]. This object-oriented mechanism works well for “distributed computing” but may not be efficient for “cooperative computing” because the intelligence is centralized in the server.

3.3 Advantages of the data based mechanism

A multi-provider and cross-network service is designed by the association of heterogeneous components from different service providers. Since new heterogeneous components will continue to hit the market, future control protocols and signaling mechanisms should be carefully designed to allow these new components to cooperate with existing ones, and to allow the initiation of the operational phase before the design of all control application has been concluded. It has been shown in [9] chapter 1 that the data based mechanism is well suited for the design of such protocols and that it will allow building a more generic interface between heterogeneous components and will provide an easier service implementation and easier cooperation between network and service providers. Such a data based mechanism has been used by management protocols like SNMP, ILMI, CMIP and NetConf [14]. However, because of the unequal management functionality distribution these protocols are implemented in a centralized architecture and it is shown in [9] chapter 9 that in a centralized configuration this *Variable Oriented* approach (data based mechanism) may be inefficient with respect to bandwidth, CPU, time and memory. Indeed the management station has to multiply in an excessive way the Get/Set comments on the various agents of the star architecture and many authors now favor a command based approach for centralized management.

Up to now, the command based mechanisms have been the only method used in signaling. Signaling protocols like SIP [15] or ISUP [16] are all based on the command based mechanism. However in the cooperative computing situation of the control plane, a single point only addresses a few entities and does not suffer from the above inconvenient of data based mechanisms for centralized architectures. This would favor the simpler and more general data based approach.

Also, in a cooperative computing environment, processes have a behavior model and a current state. Signaling information will vary with the current state of the destination process, even if the requested service is the same. Such behavior is also known by context-sensitivity as described in [17, 18], it enables a software system to adaptively take different actions in different contexts. For example if the bank call center administrator sends a fire alarm to the phone of all the employee, depending on the phone current state, the system will send a text message and a beep sound to idle phones and a voice message to off hook phones. Data based mechanism is better suited for "context-sensitivity" in control plane applications because it allows to read, with a Get request, the current state of the remote partner process and adjust to this current state by sending adapted information.

We conclude from these arguments that a command based mechanism is the advisable choice for centralized computing applications while a data based approach is a better choice for a cooperative computing application: In non centralized control plane applications data based communication is not a handicap, it offers a generic and simple interface making multi-provider and cross-network services as well as context-sensitive services easy to implement.

We therefore propose a new signaling paradigm; more adapted to the cooperative computing nature of the control plane relying on a data based mechanism. For this purpose, we define a generic data structure: the "generic context" or "GC", for the Call Instance Data of the cooperating processes and a new signaling protocol, the "*Generic Context Sharing Protocol*" (GCSP), where signaling is achieved by reading or modifying data instances in remote contexts with Get/Set/Notify commands under trust and security restrictions.

4 Generic Context overview

The generic context GC is the data structure given to CIDs in all the contexts of the participating processes. It is similar to an SNMP MIB. As a detailed description of the generic context would be too long to develop here and would justify a complete paper, we only give a small outline, preferring to focus on GCSP mechanisms in part 5.

We use the SIMPSON [19] model to organize the GC. The SIMPSON model (Signaling Model for Programmable Services over Networks) gives a structure to service and control plane sub-functions. It takes in account multi-provider services as it includes client sub-services, provider (integrator) sub-services and component sub-services. All local contexts involved in a same session have the same Generic Context data structure. A Generic Context is opened at session initiation and erased at session termination; it persists only during the service session. The data structure schema of the Generic Context is designed according to an object-oriented approach, as done in the Common Information Model (CIM) [20]. Two control applications that use GCSP communicate by a data based mechanism and there is no remote method invocation as in object oriented communication. Data may be exchanged via Get/Set/Notify commands as in SNMP [21]. Structure and relationships between data in a generic context are described using an object-oriented approach. While the SNMP MIB has a hierarchical tree view of all managed objects; the simplicity of a MIB prohibits defining more complex data and expressing relations between data elements. The Generic Context offers a richer syntax for representing control information and control objects relationships. It has an object-oriented approach to allow a greater flexibility in its design. The operations and notifications may be described at a high level of abstraction which makes standardization easier and errors less likely [22, 23]. In comparison, MIBs do not allow the same degree of reusability since they do not support inheritance and lead to the addition of duplicated schema entries as models grow up to support more vendors and more device/application types.

The Generic Context maintains at the application level a persistent communication between partner processes. For this, a process holds in its Generic Context an association, or pointer, which binds it to a partner process with which it has working relations. This association allows processes to mutually address each other; a process can send requests and notifications at any time to a partner process during a service session. Security is taken into account in the Generic Context design. A dedicated trust and security object in the Generic Context handles the authentication, access rights and ciphering issues

5 Generic Context Sharing Protocol mechanisms

5.1 Protocol overview

GCSP triggers remote operations using Get/Set commands. Instead of sending a command to make a remote process execute an action, we modify a CID in the remote Generic Context with a Set command. When the remote process detects the change, it executes the action. A prior GET may be done to know the current value but *it is not mandatory*. The prior GET may be useful for performing context-sensitive actions. For example, rather than sending a Make Call (a, b) command to a remote entity, a GET downloads the concerned part of the Generic Context of that entity, we set the “Make Call” attribute to “true”, “calling” to “a” and “called” to “b”, then upload the object to the remote context with a SET. Upon detection of the change the remote entity makes the call. A direct Set with the necessary attributes would have also made the call.

With GCSP a process can read the current state of a partner and its behavior model before taking a further action and thus can predict its future state. To respect the performance requirement of signaling GCSP should be the least verbose and should allow the use of signaling transactions. Several modifications may be done on a context before uploading it. This is equivalent to transactions in MEGACO [24] and TCAP [25] which are essential to the protocol performances [26]. Renowned mechanisms may be used to increase performance. As in an SNMP MIB, GCs objects names can be replaced by numbers to reduce the size of GCSP messages. Study [9] shows that data based mechanism allows to initiate the operational phase of services before the design of all control applications has been concluded because it is easier to enrich the protocol stack. GCSP can also be encrypted with an SSL layer if it runs on TCP. The encapsulation of objects in a Generic Context guaranties the integrity of an object and protects from unauthorized access [22, 23].

5.2 GCSP mechanisms

GCSP is a text based protocol like HTTP [27] and SIP. A GCSP frame consists of a header and a body as shown in figure 1. GCSP is an application level protocol which uses UDP as SIP does [28] or could also use the NSIS transport layer. TCP can be used to support SSL encryption and firewall traversal. Reliability is assured by timers in the GCSP protocol stack that handles messages retransmission.

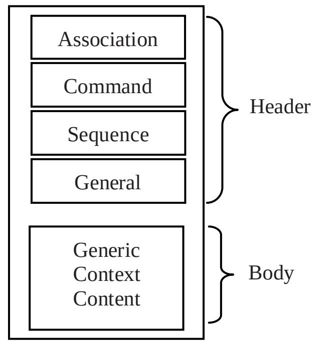

Figure 1. GCSP Frame

5.3 Protocol frame

5.3.1 Header

Header lines provide information about the request or the response, or about the objects sent in the message body. Header lines are in the usual text format, which is one line per header, of the form "header-name: value", ending with CRLF. It is the same format used for email and news postings, defined in RFC 822, section 3. We give hereafter an outline of the different sections of the header.

5.3.1.1 Association

Two GCs of partner processes are bound together with an association. A GCSP association is bidirectional; both processes can address mutually each other. The association section in GCSP header consists of the source (From) and

```
From: chahine@enst.fr 400854585532112
To: rigault@enst.fr
Source-Context: 102
Destination-Context: 53
```

Figure 2. Association section lines in a GCSP frame

destination (To) addresses (private and/or public), and the source and destination contexts IDs. Many addresses can be sent in the "From" and "To" fields. This association is similar to a TCP socket. However GCSP makes the association at the application level which allows to implement services independently from the transport protocol.
In figure 2 we give an example of association between two communicating processes. The "From" field indicates the source addresses, a public address (chahine@enst.fr) and a private address (400854585532112), while the "To" field indicates the destination address. Source-Context and Destination-Context are references of the source and the targeted GC. A context reference is unique within a single machine like a TCP port.

5.3.1.2 Commands

The command header describes the invoked command. A response is expected with a response code as in Http. Some commands have to indicate the full path of the target object in the remote GC. GCSP commands are as follow:
Get. A control process can query data in a remote Generic Context using a Get command. It must indicate the full path of the targeted part, for example the Generic Call Control part of the remote GC: Get Context.GenericCallControl lock GCSP/1.0
A *lock* keyword is mandatory if the control process wants to modify the remote context. This prevents other control processes of modifying the context before the initial control process, which made the Get command, uploads it. The lock keyword is not mandatory for read-only Get command. Following a Get command, a response is expected. The response indicates its status response. If it is 200 OK, the body of the response contains the queried Generic Context data: GCSP/1.0 200 OK
Set. A control process can upload a part of a remote Generic Context with a Set command. It must indicate the full path of the targeted part of the GC. The message body contains the Generic Context data that should be modified. The remote Generic Context is unlocked after a Set command or a time out: Set Context.GenericCallControl GCSP/1.0
Notify. GCSP is a state-full protocol. Control processes may send notifications with a Notify command. Subscription to notification is done via a Set command. For example a Detection Point DP is armed by a Set and when a filter criterion matches, a notification is sent to the concerned partner. The Notify header line indicates the object raising the notification and the message body contains the notification data. For example the notification below is sent to an application server after a filter criteria match. The GCSP body contains data relative to the Filter Criteria object which is the script ID to execute and the filter criterion priority:
Notify Context. AccessComponent.UserProfile.FilterCriteria GCSP/1.0
Open-Context. To start a process in a remote entity, an Open-Context command is sent with the Association section filled except for the Destination-Context line. A new remote process is started which opens a context and answers back with a 200 OK response and put the Source-Context in the Destination-Context line and fills the Source-Context with the reference of the new Generic Context that has been created.
Close-Context. A communication is ended with a Close-Context command. Contexts involved are closed and freed from memory. After a process receives a Close-Context command, it answers back with a 200 OK (if the Generic Context is

closed) with an embedded Close-Context command to notify the remote process that there is no more data to be send.

Describe. The skeleton of the Generic Context with its fundamental objects will be described in another paper. However to provide extensions the describe command may be used to discover the structure of a new object in a remote GC.

Lookup. Control processes that implement GCSP may communicate with other control processes using different signaling protocols via signaling mediators. A signaling mediator is a gateway that translates GCSP to another signaling protocol. GCSP may cooperate to locate the adequate mediator for a given signaling protocol.

5.3.1.3 Other headers

A sequence number tracks how many messages have been exchanged in a communication between two GCSP applications or also indicate a transaction number if any:

Sequence <sequence_number> <transaction_number>

A General header section is present in all GCSP frames. The header lines include: Content-Type, Content-Encoding, Content-Length, Date, Expiration

5.3.2 Body

A GCSP message may have a body of data sent after the header lines. If so, there may be header lines to describe the body such as: Content-Type and Content-Length. When a Get command is sent to a peer process, it usually responds by 200 OK in a header line and the queried object in the message body. Because SDP [29] does not take in account the description of an object that encapsulates another object and because the number of SDP attributes is limited to the alphabet size we use a new description language shown in the example below. In this example the objects in the figure 3 are represented as follow in the GCSP message:

```
<HEADER LINE_1>
<HEADER LINE_n>

<BLANK LINE>

#s: <Object_1>CRLF
attr_1: <value> CRLF
attr_2: <value> CRLF
attr_n: <value> CRLF
#s: <Object_2> CRLF
attr_1: <value> CRLF
attr_2: <value> CRLF
attr_n: <value> CRLF
#e: <Object_2> CRLF
#e: <Object_1> CRLF
```

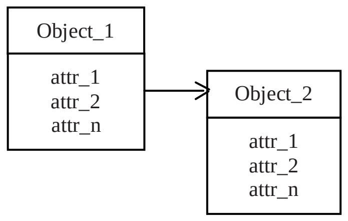

Figure 3. Generic context objects schema example

6 Related work

Data based mechanisms have been used by management protocols like SNMP, IMLI and NetConf. They are specifically designed for the centralized architecture of management and have features that do not make them usable in control applications. For example, the request ID of the SNMP header which makes an association between two entities: the NMS and the agent. This is not what is required in control softwares where associations are made between 2 or more control processes involved in a same service session. Other protocols like NetConf are hard to consider in control software for efficiency reasons since they use an XML syntax for the content body, encapsulated in RPC messages and transferred by BEEP [30], SSH or SSL over TCP. While TCP is a reliable transport protocol for management, it is a handicap for control software because of its three way handshaking at the beginning that increases the call setup delay. Also some implementation of TCP can have delays of 6 and 24 seconds in the retransmission of the initial SYN packet. These reasons made SIP go on UDP and not TCP [28] and keep TCP for firewall traversal transport-layer security protocol such as TLS [31].

The NSIS working group at the IETF has defined a set of requirements, a scenario for future signaling protocols [32], and a framework divided into a transport layer and a signaling layer [2]. The transport layer is a robust layer that will assure the transport of application signaling in a similar manner as the SS7 network does with ISUP, MAP and INAP. While most of the work on NSIS Transport Layer is accomplished, there is still a lot to be done on the signaling applications layer. GCSP could be a candidate for the NSIS Signaling Layer Protocol for Cooperative Processes.

As we will see in part 7, GCSP provides a simpler approach to the design of signaling mediators. Today’s mechanisms to accomplish multi-provider services, like web services CORBA and RMI, require that the different service components should all be on the Internet network. With the help of GCSP signaling mediators, service components may be located on different networks.

7 Validation

We have integrated the GCSP protocol to the Corebridge CTI (Computer Telephony Integration) applications suite. They consist of a server connected to a company PBX through the PBX CTI link (SIP, TAPI, CSTA or proprietary) and a set of CTI applications. A CTI application (phone bar), among many features, allows searching customers’ profiles in a database, and initiating and handling phone calls. To initiate a call, the phone bar sends a command to the Corebridge server which forwards it to the PBX. To take in account the case where PBXs are implemented as SIP proxies, we have developed a GCSP/SIP signaling mediator. This mediator receives GCSP commands from the phone bar and sends SIP commands to the SIP proxy. Reversely, it receives SIP commands from the SIP proxy and sends GCSP commands to the phone bar. This generic architecture allowed us to support a wide range of PBXs with less cost of development efforts.

8 Conclusion

In this paper, we have underlined the cooperative computing nature of the control plane software and we have reached the conclusion that “data based signaling mechanisms” are better suited than “command based signaling mechanisms” to this cooperative nature of the control plane. Thus we have provided a brief description of the Generic Context that structures the common shared contexts and we have given a detailed description of the new GCSP signaling protocol used to share and modify GCs data in the control plane. Currently we are working on a SIP/GCSP signaling mediator, future publications will give more details on this work and the design of signaling mediators to interface with the various current signaling protocols and detailed descriptions of the Generic Context.

List of acronyms

BEEP:	Block Extensible Exchange Protocol
CID:	Call Instance Data
CIM:	Common Information Model
CMIP:	Common Management Information Protocol
CORBA:	Common Object Request Broker Architecture
CSTA:	Computer Supported Telephony Applications
GC:	Generic Context
GCSP:	Generic Context Sharing Protocol
ILMI:	Interim Local Management Interface
ISUP:	ISDN User Part (SS7)
MIB:	Management Information Base
MEGACO:	MEdia GAteway COntrol protocol
NETCONF:	Network Configuration
NMS:	Network Management Station
NSIS:	Next Step In Signaling
POTS:	Plain Old Telephone Service
RMI:	Remote Method Invocation
SDP:	Session Description Protocol
SIMPSON:	Signaling Model for Programmable Services Over Networks
SIP:	Session Initiation Protocol
SNMP:	Simple Network Management Protocol
SSD:	Service Support Data
TAPI:	Telephony Application Programming Interface
TCAP:	Transaction Capability Application Part (SS7)

References

1. C. Rigault, R. Chahine, Cooperative computing in the control plane; application to NGN services and control, IFIP MAN 2005

2. IETF RFC 4080, Next Steps in Signaling (NSIS): Framework, June 2005

3. RFC 1213, Management Information Base for Network Management of TCP/IP-based internets:MIB-II, March 1991

4. Vijay K. Gurbani and Xian-He Sun, Senior Member, IEEE, Terminating Telephony Services on the Internet

5. IETF RFC 3910: The SPIRITS (Services in PSTN requesting Internet Services) Protocol, October 2004

6. IETF RFC 2458: Toward the PSTN/Internet Inter-Networking - Pre-PINT Implementations, November 1998

7. IETF RFC 2848: The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services, June 2000

8. Parlay : http://www.parlay.org

9. Aiko Pars, PHD Thesis: Network Management Architectures

10. ISO documents 9595, 9596 and ITU.700, X.711

11. W3C, Web Services Description Language (WSDL) 1.1, 15 march 2001, http://www.w3.org/TR/wsdl

12. CORBA, www.corba.org

13. William Grosso, Java RMI, O'Reilly, first Edition October 2001

14. IETF draft: http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-12.txt

15. IETF RFC 3261, Session Initiation Protocol (SIP), June 2002

16. ITU-T Recommendation Q.764, Signalling system No. 7 – ISDN user part signalling procedures, 12/1999

17. B. Schilit, N. Adams, and R. Want, Context-Aware Computing Applications, Proc. IEEE Workshop Mobile Computing Systems and Applications, pp. 85-90, 1994.

18. A.K. Dey, Understanding and Using Context, J. Personal and Ubiquitous Computing, vol. 5, no. 1, pp. 4-7, Feb. 2001

19. Astronefs, Network and Telecommunication Global Service Convergence: White paper, http://www.infres.enst.fr/~rigault/white-paper.pdf

20. Common Information Model (CIM) Standards: http://www.dmtf.org/standards/cim/

21. IETF RFC 1157, Simple Network Management Protocol (SNMP), May 1990

22. S.M. Keller: System Management Information Modelling, IEEE Communications Magazine, page 38-44, May 1993

23. W. Stallings: SNMP, SNMPv2 and CMIP – The Practical Guide to Network Management Standards, Addison Wesley

24. IETF RFC 3015, Megaco Protocol Version 1.0, Novembre 2000

25. TCAP ITU-T Q.771_Q775, TCAP: Transaction Capabilities Application Part

26. Philippe Martins, Architecture de contrôle et middleware pour les réseaux de prochaines générations et évaluation de performances, PHD thesis, ENST Paris, April 2000

27. IETF RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, June 1999

28. H. Schulzrinne, J. Rosenberg, Signaling for Internet Telephony, February 2, 1998

29. IETF RFC 2327, SDP: Session Description Protocol, April 1998

30. IETF RFC 3080, Block Extensible Exchange Protocol, March 2001

31. IETF RFC 2246, The TLS Protocol Version 1.0, January 1999

32. IETF RFC 3726, Requirements for Signaling Protocols, April 2004