WS03. Communication Abstractions for Distributed Systems

Antoine Beugnard[1], Ludger Fiege[2], Robert Filman[3], Eric Jul[4], and Salah Sadou[5]

[1] ENST-Bretagne, Brest, France, antoine.beugnard@enst-bretagne.fr
[2] University of Technology, Darmstadt, Germany fiege@gkec.tu-darmstadt.de
[3] RIACS/NASA Ames Research Center, USA rfilman@mail.arc.nasa.gov
[4] University of Copenhagen, Copenhagen, Denmark eric@diku.dk
[5] Université de Bretagne Sud, Vannes, France sadou@iu-vannes.fr

Abstract. Communication is the foundation of many systems. Understanding communication is a key to building a better understanding of the interaction of software entities such as objects, components, and aspects. This workshop was an opportunity to exchange points of view on many facets of communication and interaction. The workshop was divided in two parts: the first dedicated to the presentation of eight position papers, and the second to the selection and discussion of three critical topics in the communication abstraction domain.

1 Introduction

As applications become increasingly distributed and as networks provide more connection facilities, communication takes an increasingly central role in modern software systems. Many different techniques and concepts have been proposed, in both research and industry, for providing structure to the problems of communication in software systems. Over the last 15 years, the basic building blocks for distributed object systems have emerged: distributed objects, communicating with Remote Message Send (RMS), also known as Remote Method Invocation (RMI) or Location-Independent Invocation (LII). Message-oriented middleware has also seen some clever implementations. However, it has also become clear that while such abstractions are by themselves sufficient to expose the hard problems of distributed computing, they do not solve them.

Databases and graphical user interfaces are examples of software elements that were once difficult to program but have, through the right abstractions and implementations, been greatly simplified. The question central to this workshop is whether some similar abstractions and can be devised for simplifying distributed applications. Just as successful database components required not only mechanisms for storing and retrieving data, but also abstractions like query languages and transactions, successful communication abstractions will require more than the mere ability to communicate.

At previous ECOOP workshops on *Communication Abstractions*, we identified some features (security, reliability, quality of service, run-time evolution,

causality) that are central to any communication abstraction and some particular communication abstractions such as Peer-to-Peer or Publish/Subscribe. The goal of this workshop was to work on the definition of new and better communication abstractions and on the distributed-specific features mentioned above.

We received 17 positions papers. Three were related to cryptography only and were considered out of scope. All others were reviewed by at least two members of the program committee and 8 were considered bringing an interesting point of view and deserving a chance to be discussed.

We organized the workshop as follows:

- The morning was dedicated to short, 15 minute presentations of selected papers. The paper authors entertained questions from the workshop attendees and provided clarifying responses.
- In the afternoon, we formed three working groups for deeper discussion of particular issues in communication abstractions. The group reported their conclusions to the collected workshop at the end of the day.

2 Position papers abstracts and discussions

2.1 Event Based Systems as Adaptative Middleware Platforms [7]

Adaptive middleware is increasingly being used to provide applications with the ability to adapt to changes such as software evolution, fault tolerance, autonomic behavior, or mobility. It is only by supporting adaptation to such changes that these applications will become truly dependable. In this paper we discuss the use of event based systems as a platform for developing adaptive middleware. Events have the advantage of supporting loosely coupled architectures, which raises the possibility of orthogonally extending applications with the ability to communicate through events. We then use this ability as a way to change the behavior of applications at run time in order to implement the required adaptations. In the paper we briefly describe the mechanisms underlying our approach and show how the resulting system provides a very flexible and powerful platform in a wide range of adaptation scenarios.

Questions and Answers

How do you know where to put the hooks on the application? We assume we have access to the source code of the applications and know how to generate the point cuts that are needed. In some cases it is not necessary to have access to the source code as we can use PROSE to trap generic operations like outgoing socket connection calls to a certain location. Nevertheless, we are right now assuming that the source code or at least the relevant parts of the source code are known.

Centralized vs. distributed We currently use an approach similar to Bluetooth and we pre-allocate the master. We have not yet done performance studies on the differences between centralized and distributed event management.

Only application extensions or middleware extensions as well? We can do both. Certainly the application. The event system must run in Java and be amenable to PROSE but in principle it is possible to extend the event system as well.

What about the Reliability model? Right now we do not have reliability. We were trying to prove the concept of dynamic adaptability using events and we have not yet gone as far as considering reliability. At this stage we are focused on basic performance problems inherent to wireless networks. Once those are solved, we will then add additional layers of functionality such as reliability, security, etc.

Without reliability is the system usable? No. But what we have is only the first step. We have not yet completed the system and there are several layers missing to make it a realistic platform. Reliability will be implemented at a higher level than what has been described in the talk.

Removing and introducing new functionality, cross interactions We have two type of extensions, a non-aspect extensions (module replacement) and an aspect based extension that supports further extensions. The assumption is that the person introduces extensions knows what she is doing. Extensions are part of the code of an application and one can create bugs much as bugs are introduced in normal code.

What devices do you use? Laptops in a first attempt. IPAQs with Linux and Java right now.

How do you map new events to the parameters and the actual application? PROSE uses the JVMDI to extract the parameters of calls and we use that information to cast parameters as necessary. For matching parameters, we use a simple algorithm that simply does pattern matching. Future versions will do a more complete job in this area.

2.2 Interaction systems design and the protocol and middleware centered paradigms in distributed application development [2]

This paper aims at demonstrating the benefits and importance of interaction systems design in the development of distributed applications. We position interaction systems design with respect to two paradigms that have influenced the design of distributed applications: the middleware centered and the protocol-centered paradigm. We argue that interaction systems that support application-level interactions should be explicitly designed, using the externally observable behavior of the interaction system as a starting point in interaction systems design. This practice has two main benefits: to promote a systematic design method, in which the correctness of the design of an interaction system can be assessed against its service specification; and, to shield the design of application parts that use the interaction system from choices in the design of the supporting interaction system.

Questions and Answers

What is the notion of platform-independence you adopt? The application part is defined at a high level of platform independence, which means that the applications parts are defined in a way that is unconstrained by potential target platforms. This is because they are defined in terms of the external behaviour of the interaction system.

The design of the interaction system itself may rely on an abstract platform, which will define the notion of platform-independence explicitly, by defining what characteristics of potential platforms are to be considered.

Can there multiple interaction systems coexist? What if they interact? Yes, they can coexist. There is a duality between interaction systems and system parts. If a system part interacts with two or more interaction parts, it can be considered itself an interaction system.

How do you compare your approach with approaches in coordination languages, EAI or MDA? This comparison is the next step of this approach. We have a set of design concepts that have been derived from LOTOS. The group I work on was involved in the development of LOTOS and departed from it for a more expressive set of basic design concepts based on interactions and relations between interactions. As future work, we should look at other coordination languages and see how the basic concepts can be mapped[1].

Is the boundary between two systems is itself an interaction system? Yes, and it should be considered explicitly as a interaction system if it is interesting according to the criteria I've presented for justifying interaction system design. This is possible in the approach because of the recursive definition of interactions systems.

Where does one stop? (because otherwise you'll have infinite interaction systems) We should stop when the interactions are simple enough to be realized at implementation. This is arbitrary, again according to the criteria to justify interaction system design.

How does the boundary between system parts look like? How to define it? We have a set of basic design concepts for that. Interaction points are abstractions of the mechanisms shared by two system parts for interworking. Interactions occur at interaction points. Interactions are shared activities. They may take time and occur synchronously, in that all system parts involved in the interaction perceive the occurrence at the same time when the interaction completes (and have the same perspective on that). The use of synchronous interactions allow us to model tightly coupled interactions at a very high level of abstraction (without considering implementations of the synchronous interactions)[2].

Do system parts have to know (be aware of) the interaction points? This is a basic set of design concepts that allows a mapping to different implementations solutions. The interaction points is an abstraction of the mechanisms shared by two system parts.

[1] with respect to the relation to MDA please have a look at [10]
[2] please have a look at reference [1]

2.3 A Modular QoS-enabled Load Management Framework for Component-Based Middleware [4]

Services are increasingly dependent on distributed computer systems for office workstation support, banking transactions, supermarket stock supply, mobile phones, web services etc. Web services are being increasingly used for e-commerce, education and for information access within and between organizations. This leads to increasing need for high performance, enterprise level, distributed systems. The most suitable way to achieve high performance is by using load management systems. Delivering end-to-end QoS for diverse classes of applications continues to be a significant research area. There are individual technologies, based on prior research, generally targeting only specific domains and usage patterns. This paper presents a new QoS-enabled load management framework for component oriented middleware. It proposes a new approach for load management and for delivering end-to-end QoS services. The proposed framework offers the possibility of selecting the optimal load distribution algorithms and changing the load metrics at runtime. The QoS service level agreements are made at user level, the application being managed not being aware of this. Work is in progress for creating a simulation model for the proposed framework and for evaluating the performance improvements it offers, with the current focus on the enterprize java beans platform.

2.4 Communication Abstractions in MundoCore [6]

Ubiquitous computing with its many different devices, operating system platforms and network technologies, poses new challenges to communication middleware. Mobile devices providing vastly different capabilities need to integrate into heterogeneous environments. We have identified a key set of requirements for ubiquitous computing communication middleware which are modularity, small footprint, and ability to cope with handovers. In this paper we present MundoCore which provides a set of communication abstractions based on Publish/ Subscribe for ubiquitous computing scenarios. We present the communication model of MundoCore and discuss our implementation, along with our experiences and observations from the implementation process.

Questions and Answers

What is currently implemented? MundoCore on C++ and Java. IP-based transport services with optional AES-128 encryption. Routing service. Event routing service. Remote method calls based on own precompiler. Language extensions based on own precompiler. Current Limitations: Currently only SOAP message format supported, Event filters simplistic, No access services (DUN)

Could it be ported to an Ericson Smartphone? Yes. The footprint of the C++ version is below 100KB. We plan to do a port on Nokia Series 60 (EPOC) in the near future.

How does 'emits' behave in case of inheritance? Emits is implemented by means of protected static inner classes. If the enclosing class is derived, the emitter-class is also derived. This way, inheritance is fully supported by using standard Java language features.

2.5 CSCWGroup: A Group Communication System for Cooperative Work in the WAN [9]

Group communication is an important part of CSCW system. Originally its studies focused on LAN environment, and later on WAN for users concentratedly in several areas. However, it was rarely discussed for group members distributing dispersedly. For widely supporting the group communication in the WAN, an new group communication system– CSCWGroup is built. The client/server cluster architecture is introduced in our system. Based on it, a set of management mechanisms is designed in this paper. For evaluating the system performance, simulation software JAVASIM is used to simulate the running process of the system. With the measurement results of message delay and system throughput, it is proven that the system has the good scalability and can well support the group communication in the WAN environment.

2.6 Adam: a library of agreement components for reliable distributed programming [8]

This paper presents ADAM, a component-based library of agreement abstractions, used to build reliable programming toolkits. ADAM is based on a generic agreement component which can be configured to build many abstractions such as *Atomic Broadcast*, *Membership Management*, *View Synchrony*, *Non-Blocking Atomic Commitment*, etc. Currently, ADAM is used by the Eden framework, which is a group-based toolkit and also by OpenEden, which is an implementation of the Fault-Tolerant CORBA specification.

Questions and Answers

Is there any relation with other group communication system? We put the emphasis on a component approach and try to implement various agreement problem by configuring parameters.

How F, GET are provided? A concrete agreement component registers and offers the appropriate functions.

2.7 Anonymous Communication in Practice [3]

Anonymity is something most people expect in their daily lives. We usually expect to be anonymous when we vote, and, for example it is essential for many telephone hotlines for people with serious problems that the users are anonymous. In general there are many daily scenarios where people would like to

communicate with others without letting any third party know who they are communicating with. Communication on the Internet is not anonymous, but if we want our activity online to be as private as offline, anonymity is needed. The goal of this paper is to provide a view on the field of anonymous communication with a focus on practical solutions to aid the designers of distributed systems where privacy is an issue. We explain the problem of anonymous communication and discuss socalled rerouting based solutions. Our discussion includes an overview of security issues and a survey of useful designs.

Questions and Answers

Is dummy traffic sometimes used to hide communication? Yes, dummy traffic is needed to prevent traffic analyzing. Tarzan uses pairs of rerouters that always sends the same amount of traffic. If no real traffic is available, random data is sent.

Does mixing scale? The short answer for most designs are yes. If the rerouters are organized in a decentralized way, it would not be a problem with lots of servers. The number of rerouters must however grow with the usage otherwise the system will obviously slow down, because the work load will increase.

How could the rerouters be organized? In theory they could be placed in one room, or we could even make a system with just one rerouter—in practice that would be to vulnerable. It would be good to spread out rerouters geographically as well as on different "groups" (nations, companies, organizations) to make the rerouting hard to attack by compromising rerouters.

What if compromised rerouters changes packet? One attack could be just to drop packets, this cannot be prevented. Another attack is to change the content before resending the packet but otherwise each packet should be integrity protected. This is actually easy to do, because often there is a layer of encryption for each rerouter, a checksum could be included before encryption, this could be verified after decryption.

How does "receiver" anonymity work? Usually it is done by the receiver providing information on a path in the rerouting network, the sender must obtain this before a communication could be initiated.

This needs some kind of Publish-Subscriber system!? Yes.

2.8 Communication Abstractions Through New Language Concepts

In this paper [5] we take the position that dedicated language concepts are to be considered as the solution for introducing commonly used communication abstractions into distributed programs. In our research we explicitly abandon middleware solutions, such as generation of stubs and skeletons. They do not give rise to the new ways of thinking that will be required for the construction of distributed and mobile systems in highly dynamic environments such as interconnected desktops, pdas and domotics. More specifically, we think that

both the complexity and weakness of most middleware technology and the solutions the spawn is due to the fact that the technology is statically typed and class-based. Indeed, the major raison detre for generated interfaces and stubs is to satisfy type checkers for static languages. Because of that, we are starting to investigate how we can put the properties of prototype-based languages to structure and simplify the development of mobile agent software and thus also distributed systems.

In this paper we introduce some preliminary resulting communication abstractions based on the delegation mechanism most prototype-based languages feature. As a concrete case we will discuss some coordination problems in the master-slave design pattern and do a few *gedankenexperiments* in language design in this context.

Questions and Answers

Can class and prototype based languages be mixed? The prototype based language we are designing also mixes both concepts to avoid some of the problems associated with prototype based languages (cfr. Paper on Intersecting classes and prototypes on http://www.dedecker.net/jessie/publications), such as the usage of mixins to solve the problem of encapsulation with object inheritance. However, we do not have looked for a static type system while maintaining sufficient dynamicity of the program.

What are the semantics when the WE-construct is used in a child? When the WE construct is used in a child, then the message is sent to the children of that child. We are also exploring the possibility of adding another keyword that is the inverse of the super, that is to send messages to the direct children of a parent as opposed to sending the message to all children in the tree. Probably more clearer names should be chosen for these keywords.

Why introduce new language concepts and not implement them through the data abstractions that are already present in the language? There are several advantages of having dedicated language concepts that support the expression of communication abstractions: First, it can impose a new way of thinking to the developer. Second, it can influence the design of the program so it can aid in a better design of the program leading to a better reusability and understandability of the code. Third, compilation techniques can be used to optimize the use of these constructs depending on the context where they are used. However, we need to be very careful in our choice of what constructs we choose to add to our language. The constructs that are added to the language should be as orthogonal as possible with clearly defined semantics.

3 Discussions

The second part of the workshop was dedicated to discussions and working groups. After a brainstorming session where attendees suggested several subjects of discussion, we selected three of them for further exploration: communication abstraction taxonomies, quality of service, and peer-to-peer architectures.

3.1 Communication Abstraction Taxonomy

The "taxonomy of communication abstractions" working group was an attempt to (in an hour and a half) to organize the space of communication abstractions. The papers in the communications abstractions workshops cover a wide range of topics. This working group was an attempt to bring some regularity and organization to the communication abstraction space. Identification of the dimensions of the communications space has the additional advantage that many possibilities for communication organizations become evident by their place in the space. If, in some sense, a communication abstraction has dimensions such as "synchronicity," "object identity," and so forth, and the possible choices for each dimension can be identified, then the space itself defines a wide variety of models to be examined and characterized.

Communication abstractions are about modelling concurrent processing systems. Models serve as a foundation for analysis. By examining a model of communication, one can prove properties such as reliability, security, or nontermination. By analyzing the behavior of a model, one can derive metrics such as the temporal, space, or message efficiency of algorithms.

One thing that the discussion quickly revealed was that communication abstractions vary in their degree of detail and depth. By considering or defining additional features about a model, additional facts about that model can be derived. However, superfluous detail gets in the way of analysis of more fundamental properties. To reflect these distinctions, the group defined a "chemical hierarchy" of communication abstractions: the most basic models were the "quarks" of communication, assembled and extended successively into "atomic," "(simple) inorganic" modules, and most complexly, "organic" molecules.

Quark dimensions include:

Entities This speaks to the kinds of things communicate. Most familiar object identity is communicants as discrete objects with identity.

Population dynamics. This dimension is the lifecycle of communicating entities. Examples of lifecycle issues include whether entities can be created or destroyed in the model.

The "atomic" dimensions speak to the primitive communication model. Subelements of this topic include

Syncronicity Whether the model supports synchronous or asynchronous communications (blocking and unblocking)

Sharing Whether communication is by message passing or shared memory

Casting Whether communication is monocast (directed at a particular recipient), multicast (directed at a defined set of recipients) or broadcast (receivable by everyone)

Signatures Whether communications communications identify the sender.

Moving up the complexity chain, we can build communication models with the complexity of simple molecules. Molecular communication models add additional elements with respect to dimensions such as:

Failure That is, what guarantees does the system make about communication delivery and what notification the system provides for communication failures. Failure notions also include consistency models for shared memory.

Arbitration Who sees things in which order. A communication model might hypothesize, for example, that messages are received in the order sent or merely that a sent message will be eventually received.

Fairness Does the model make any assertion about order of processing.

Structure of messages What structure (for example, required or allowed fields, varieties of message types) does the model demand for message content

Direction of flow of information In the communication act, does information flow from the initiator of communication to a target, from the target to the initiator, in both directions, or does no more information than synchronization get exchanged?

Streaming Are messages discrete entities, or does the model support some form of "streamed" communications.

Coordination What mechanisms does the model hypothesize for synchronization, coordination, or serialization among entities. For example, a model may hypothesize recognition of the "termination" of a set of "objects."

Model builders can develop models with organic complexity by embellishing models with additional features. These include:

Locus A model may support a notion of physical locality. This may be a simple as distinguishing between elements that are "local" versus ones that are "remote," or might extend to an entire proximity structure. At the use level, this distinction may be seen in "access transparency" and "location transparency." Having introduced a notion of locus, we can also address mobility: the ability of an object to change its locus.

Connectors A model might divide its entities into "communicators" and "connectors," or even make other distinctions in the entity space, with different actions and privileges ascribed to different kinds of entities.

Annotation A model might provide additional meta-information about the communicating entities. Examples of such information include the protocols (interfaces) the entity supports, or its "ownership."

Conversations A model may provide "conversational" mechanisms that support a pattern of communications. Examples of such conversational mechanisms include transactions and protocols.

Quality of service A model may provide mechanisms for varying the quality of service to different entities.

Security Having defined communication and message structures, a model may address properties such as guarantees of information reaching its destination, data integrity, anonymicity, eavesdropping, and identity spoofing.

Time All the of the above issues may be extended with temporal notions, including, for example, the ability to put temporal limits or exceptions on other behaviors.

We note that the above discussion is shadowed by a kind of "Turing equivalence theorem." The distinctions are in some sense arbitrary, because choices in many dimensions may be used (in a sufficiently rich computational environment) to mimic the behavior of the opposite choices. For example, shared memory can be understood as a particular variety of messages to a memory entity; asynchronous communication can be mimicked using synchronous communication and secretary threads.

3.2 Quality of Service

One of the topics discussed during the workshop was the relationship between quality of service requirements, the separation of concerns, and communication abstractions. Obviously, engineers will benefit from abstractions that can be refined and whose quality of service can be customized to the needs of the application or the deployment environment. The central issues are where and when the necessary QoS parameters are determined, and how their provision is reflected in the communication abstractions.

A separation of concerns is reasonable in several respects. We should assume multiple layers of abstraction due to their interrelated but different focus: an application, a middleware, and a protocol layer. This facilitates simple, not too general communication abstractions that are focused on a specific layer. Different techniques may be applied on the abstractions to handle QoS requirements. For instance, aspect-oriented programming (AOP) and filter composition are readily available for application programming, while their applicability and relationship to reflective middleware and interceptors in the middleware layer is an open issue. On protocol and operating system layers even other approaches are expedient, cf. the x-Kernel.

A clean separation of QoS management from communication abstractions seems to have advantages, but cannot be achieved always and may even be undesirable. While leaving abstractions as simple as possible is reasonable, one may want QoS handling to be part of abstraction primitives/APIs for two reasons: to specify application requirements and for awareness of QoS degradation. The latter is important to enable an application to react to scarce resources, which mainly determines whether QoS should be integrated into the abstractions. Is the component itself not affected by QoS degradation or has it to know of the available QoS? At design time, compile/deployment time, or at runtime? Thus, depending on the availability of the afore-mentioned techniques to implement QoS handling in the different layers a number of different approaches for QoS-aware communication abstractions are conceivable, ranging from simple primitives that are implemented in different ways (e.g., send() methods that use either simple TCP or atomic broadcast services), to QoS aspects woven into application code wrapping the actual abstraction primitives, to QoS specifications that are traded and matched to existing services.

3.3 Peer-to-Peer

This discussion group may be seen as a continuation of the one of the last year, which was concluded as follows:

Peer-to-Peer (P2P) systems promote technological and social changes like new devices, new way of working, going both ways. P2P as a new communication patterns demand new solutions such as fault tolerant TCP/IP and no central authority. This kind of communication, which may ne anonymous or pseudonymous, leads to awareness, heterogeneous devices and more freedom ("cutting out the middle man").

Right now there is no consensus on the definition of P2P communication. All proposed definitions reflect its use rather than its concepts. Then our group focused its discussion on the concepts behind P2P communication. Here is the list of the concepts which we identified:

- no centralized control: This often leads to a greater system robustness.
- A symmetrical communication, which contrasts with the Client/Server model's asymmetrical communication.
- Nodes are servers and clients at the same time:
  - A peer takes the server role when it receives a client request,
  - A peer takes client role on external events, because the client initiates communication to servers,
- Peers are equal partners at the same level. There is no hierarchy between peers.

Another point was discussed by the group, concerns which abstractions are needed for P2P communication. Abstraction may relate to the following aspects:

- Mapping: how to map the peer ID to its location and how are IDs assigned.
- Peer discovery: how to discover other peers and negotiate communication channels.
- Asynchronous and event-driven communication: Several communications models may be taken as example. Among them we can quote broadcast (one to all), multicast (one-to-many) and publish/subscribe communication model.

These points can be taken as a starting point for future research.

4 Workshop Conclusions

Understanding and categorizing interactions among software entities appears to be a crucial in the future. This workshop attempts to gather people from various communities that are working on the same problem in order to meet and share their points of views. Communication is essential to multi-agents systems, software architecture, and distributed systems, but these fields have different goals.

This year, the workshop was more oriented towards platforms and implementation and adaptation of communication means.

Finally, we intend to setup a mailing list and a collaborative WikiWiki site in order to share ideas and projects[3].

The attendees suggested organizing a follow up workshop next year.

References

[1] Joo Paulo Almeida, Marten van Sinderen, Lus Ferreira Pires, and Dick Quartel. A systematic approach to platform-independent design based on the service concept. to appear Sept. 2003.

[2] J.-P. Almeida, M. van Sinderen, D. Quartel, and L. Ferreira Pires. Interaction systems design and the protocol- and middleware-centred paradigms in distributed application development. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[3] C. Boesgaard. Anonymous communication in practice. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[4] O. Ciuhandu and J. Murphy. A modular qos-enabled load management framework for component-based middleware. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[5] J. Dedecker and W. De Meuter. Communication abstractions through new language concepts. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[6] J. Kangasharju E. Aitenbichler. Communication abstractions in mundocore. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[7] A. Frei, A. Popovici, and G. Alonso. Event based systems as adaptative middleware platforms. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[8] F. Greve, M. Hurfin, J.-P. Le Narzul, Xiaoyung Ma, and F. Tronel. Adam: A library of agreement component for reliable distributed programming. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[9] Yang Jia and Jizhou Sun. Cscwgroup: A group communication system for cooperative work in the wan. http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/2123258, July 2003.

[10] D.A.C. Quartel, L. Ferreira Pires, M. van Sinderen, H.M. Franken, and C.A. Vissers. On the role of basic design concepts in behaviour structuring. *Computer Networks and ISDN Systems*, 29:413–436, 1997.

[3] If you want more information please contact antoine.beugnard@enst-bretagne.fr