Designing a Configurable Group Service with Agreement Components*

Fabíola Gonçalves Pereira Greve[1] **, Jean-Pierre Le Narzul**[2]

[1]Departamento de Ciência da Computação (DCC)
Universidade Federal da Bahia (UFBA)
Campus de Ondina, 40170-110 Bahia, Brasil

[2]GET/ENST Bretagne and IRISA - Adept
Rue de la Chataîgneraie - CS 17607
35576 Cesson-Sévigné Cedex, France

fabiola@ufba.br, JP.LeNarzul@enst-bretagne.fr

Abstract. *In the recent past years, many group toolkits, providing a support for the construction of reliable applications, have been designed. Even if the goals of their designers was similar, these toolkits differ in (i) the way the problems are tackled and (ii) the way the protocols are structured and set up in real systems. This paper presents the underlying design principles of a group system. It follows two innovative approaches which contribute to its flexible and configurable character. From an algorithmic point of view, the group primitives are implemented as instances of a generic consensus service. This choice leads to a great number of advantages: (a) the computation in the group is guaranteed as soon as a quorum of entities can communicate, (b) decoupling the group membership service from the communication service, (c) a better control of the dysfunctions in periods of strong network instability. From an architectural point of view, the elementary group protocols are regarded as autonomous agreement components, which can be combined freely for the implementation of other richer reliable services. This strategy differs from the classical ones in which the protocols are structured according to a fixed hierarchy of classes following a stack-based pattern of interaction.*

1. Introduction

For many years, the group paradigm has been recognized as a very useful abstraction for building distributed applications in various domains: cooperative work, teleconference, distributed games, etc. The group paradigm has proven also to be a natural candidate for designing fault-tolerant applications, above unreliable settings, by replicating a service in several objects located at different hosts. When used for replication, the group, represented by a collection of objects, is seen by the clients as a single logical entity; it is addressed transparently by the clients that are not aware of its composition.

Of course, the group composition can evolve along the time (members can join it, others can either leave it or crash) just as its state. The essential requirement for the survival of the group (as being a reliable object) is the common share of the same vision of events (changes on

*This work is partially supported by CNPQ/Brazil grant number 40.80.86/03-2.

composition, delivery of messages, crashes) by all its members; that requires the coordination of its actions by the means of agreement algorithms.

Given the importance of the group model, many efforts were carried out during this last decade to understand the problems related to the implementation of such a paradigm. These efforts led to various theoretical results and the development of many platforms. The set of works realized at the Cornell university – ISIS [1], HORUS[2], ENSEMBLE[3] and SPINGLASS[4] – are one of the most significant examples. It is important to notice that the statement of meaningful specifications, as well as the design and implementation of a group service are far from being commonplace [5, 6]. These difficulties are due mainly to the impossibility results regarding the agreement problems to which one is confronted during the implementation of the functionalities of the paradigm in asynchronous environments [7, 8].

One notices that various practical systems have the disadvantage of not specifying under which conditions the properties which they are supposed to implement are assured [5]. This is essential, considering the negative results regarding the resolution of agreement problems. It is thus necessary to clarify which is the behavior of the system when the assumptions established to guarantee its termination are not satisfied. The use of the consensus as building block for the construction of group services provides a concrete response to this need [9]. The precise characterization of the liveness and safety properties ensured by these solutions allows an exhaustive control of the behavior of the applications in the occurrence of network dysfunctions. Besides, theoretical solutions founded on the consensus support the construction of well structured protocols [10]. Our interest in this work is to benefit from the modular nature of the consensus based solutions to build extensible and adaptable systems.

The great majority of the group toolkits adopt a layer-based way of interaction between the elementary group protocols [2, 11]. If, on one hand, the structure imposed by a stack-based organization defines a way of developing an abstraction which is clear and easy to understand, on the other hand, the need to accomodate the functionality of the abstractions in layers that can only communicate with their adjacent layers imposes a very restrictive design model. Moreover, the flexibility of a stack-based architecture is rather limited and it does not allow for a real adaptation to the application needs or to the various qualities of the communication medium. Driven by these considerations, we address in this article a solution based on the use of the component technology. We believe that the properties of reusability, configurability and composability exhibited by this technology is of great interest for building a library of flexible reliable abstractions.

This paper is dedicated to a presentation of the underlying design principles of a component-based group system called EDEN [12]. The design is driven by the recent theoretical results regarding the realization of an agreement in an asynchronous environment. Moreover, it advocates the use of the component technology to struture the protocol classes. In such an approach, a reliable application will be designed as the composition of several independent components, connected via a communication platform and cooperating by well-defined interfaces. In EDEN, these components are part of the ADAM [13, 14] library and are structured using EVA [15], an event based architecture.

The paper is composed of five sections. Section 2 characterizes a group service. Section 3 presents briefly some of the group toolkits. Then, Section 4, the core of this work, discusses the many design alternatives proposed previously and compares them with the EDEN issues. Section 5 presents EDEN. Finally, Section 6 concludes the paper.

2. Group Service

A group is a collection of related processes, considered as a single logical entity. This paradigm covers two basic services, namely, the *group membership* service and the *group communication* service.

Group Membership. The group membership is in charge of providing, to the processes members, the current composition of the group. This is a major attribute of the state of the group (the other attributes of the state of the group are related to the application service or computation which the group is supposed to provide). The composition of the group evolves according to the will of the processes to join it or to leave it, and to the occurrence of processe crashes or communication channel failures. The current group composition is usually named a view.

The group membership problem has been introduced and solved for the first time by Cristian [16] in the context of synchronous distributed systems. The work on the membership problem in asynchronous systems has been pioneered by the Isis system [17]. Unfortunately, its specification was incomplete [5]. The asynchronous group membership problem was later proved to be impossible to solve without additional assumptions (on the detection of crashed processes) [8]. The existing difficulties to specify and solve this problem are directly related to the impossibility result regarding reliable detection of the failures in the asynchronous model [18, 7]. Since real failures combined with eventual wrong suspicions can carry out to a division of the group in sub-groups, two approaches for the management of the composition appeared: *primary partition systems* and *partitionable systems*.

Primary Partition x Partitionable Systems. A primary component membership service ensures that at any time the group is implemented by a single view. From a user perspective, it means that the membership service ensures the total ordering on the set of the views. A partitionable membership service allows different views of the same group to coexist (concurrent views). This means that the set of views perceived by the user is partially ordered. In that case, the processes of each view behave as if they were the only ones that are currently implementing the group. In each one of these components, the service (or at least a part of this one) must continue to be assured. Possibly, when the communication is restored, the group membership service carries out a mechanism of fusion of views in order to restore the group in a consistent state (taking into account the various states of the broken components).

Group Communication. The aim of a group communication service is to provide application processes with communication primitives that are well-suited to the group computing paradigm. The main primitive offered by this service is a *reliable multicast* facility allowing a process to send messages to all group members in an atomic manner. Usually, ordering guarantees (e.g., total order, causal order, fifo order) are associated with this multicast primitive [17]. This originates a number of other primitives. The fundamental ones for the implementation of a replicated service are (i) total order broadcast (also known as *atomic broadcast*), which ensures that all messages are delivered in the same order by all the group members and (ii) *view synchronous multicast* which aims at coordinating message delivery and installation of new views.

2.1. Group Protocols as Agreement Problems

Most of the reliable distributed abstractions of interest in group computing (atomic broadcast, membership management, view synchrony, leader election, ...) are agreement problems. With regard to all these problems, processes belonging to a same group have, from time to time, to

reach an unanimous decision. For example, in the atomic broadcast problem, due to the fact that messages are not received in the same order by all the processes, determining the delivery order can be viewed as an agreement problem. To construct this common global knowledge, all (non-crashed) members of the group have to repeatedly reach new agreements to unanimously order the new arrived messages.

The Consensus Problem. The agreement problems family can be characterized by a single abstraction, namely the *consensus* problem. Informally, this one can be defined in the following way. Each process proposes a value and all correct processes have to decide the same value, which has to be one of the proposed values. Consensus is the "greatest common denominator" among agreement problems and this is practically and theoretically very important. From a practical point of view, this means that any solution to consensus can be used as a building block on top of which solutions to particular agreement problems can be designed. From a theoretical point of view, this means that an agreement problem cannot be solved in systems where consensus cannot be solved.

Unfortunately, the consensus problem is actually impossible to solve in a deterministic way in asynchronous distributed systems when even a single process may crash [7]. Fortunately, to circumvent this negative result, several approaches have been investigated. One of them is based on the concept of unreliable failure detectors proposed by Chandra and Toueg [9]. The role of a failure detector service is to maintain a list of correct processes and suspected processes. Of course, due to the asynchronism of the system, it is impossible to write perfect failure detectors. But, if we consider that (i) any process that crashes is eventually suspected (called "completness property"), (ii) there is a time after which there is a correct process that is no longer suspected (called "accuracy property"), and (iii) a majority of processes within the group never fails, then it is possible to solve the consensus and some other agreement problems.

3. Group Toolkits

In a recent past (ten years), many group toolkits have been implemented. All of them allow to build reliable applications based on the group paradigm. We have classified these toolkits in families, which correspond more or less to the research laboratories where the projects were carried out.

- Isis [1], Horus [2], Ensemble [3] and Spinglass [4] from Cornell university. These systems have been used in the following toolkits, Orbix, Electra [19], AQuA [20] for a fault-tolerant Corba implementation.
- Totem [21] from the California university of Santa Barbara and Transis [22] from Hebrew university, Jerusalem.
- Phoenix [23] , Bast [11], OGS [24] from EPFL, Lausanne.
- Consul [25], Coyotte [26], Cactus [27] from Illinois university, Urbana.
- EDEN [12] and ADAM [28] developed conjointly by INRIA-IRISA, ENST Bretagne at Rennes and UFBA, Salvador.

The first group system generations (Isis, Totem, Transis, Consul) especially brought initial answers to the problems arising when reliability must be ensured in an asynchronous environment. Thereafter, the majority of them evolved to more modular and flexible versions (HORUS, Ensemble, Coyotte, GARF, BAST) providing full dependable issues (fault-torerance, security and real-time constraints). Further versions fulfill better the requirements of the modern applications: distributed objects support, deployment in large scale, adaptability, mobility (Spinglass, AQuA, ETERNAL, OGS, Cactus).

4. Design Choices

Even if the designers of the group systems originally had the same goal, their toolkits differ in a considerable manner (i) in the way problems are tackled and (ii) in the structuration of the protocols. In the rest of this section, we will describe some of the main characteristics of these systems in order to justify the design choices used in the development of EDEN. We are in particular interested by (i) their behavior in face of false suspicions and (ii) the way their protocols are structured.

4.1. How to React to False Suspicions

Previously, we discussed the impossibility of detecting with precision process crashes. In case of a scenario in which erroneous suspicions remain, two main approaches for the progression of computation in the group are distinguished. In a first strategy, named *progression by safe group*, the computation within the group requires the participation of all correct processes. In a second strategy, named *progression by long-lived group*, the evolution of computation requires only the participation of a subset of the correct processes (a quorum). Let us explain the characteristics and implications of each one of them.

Evolution by Safe Group The approach by safe group consists in keeping only correct processes inside the group. If some process is suspected to be faulty, it should be removed from the group view in order for the computation to take place. It means that the management of the communication depends on the management of the group membership and that it must trust the failure detector. To avoid execution blocking, the information generated by the latter is used to remove from the group view the processes suspected to be faulty, and this even if such a suspicion proves to be abusive. In order to avoid too frequent errors, the suspicion timeouts associated with the failure detectors must be well tuned.

To our knowledge, all the partitionable systems follow this strategy: Totem, Transis, Horus and their derivatives. In this context, the consequences of erroneous suspicions could be the bursting of the group, its partitioning in several minority components or in the worst case, the loss of the primary component. In the case of the Isis primary partition system, since the activity of the processes expelled from the group is stopped, this can carry out to a collective suicide.

Evolution by Long-Lived Group The approach by long-lived group is less conventional and more innovative. It consists in decoupling, as much as possible, the services provided by the group and authorizing, as soon as possible, the progression of computation even in the presence of failures.

The use of failure detectors in the design of agreement protocols enables such an approach. The adoption of the theoretical solutions founded on unreliable failure detectors authorizes the evolution of computation as soon as a majority of correct processes are able to communicate. Thus, as soon as the communication between the partitions is established (real or virtual partitions are they) and the contact between a quorum of processes is re-established, computation progresses.

One can thus benefit from the advantages related to the use of the consensus and the unreliable failure detectors as building blocks to authorize decisions even if the group is not in a safe state. Message ordering takes place even in the occurence of suspicions. View changes are carried out only with a majority of participants. This model allows the independent implementation of group protocols. The failure detectors timeouts can be adjusted in order (i) to speed the decisions up, which is appropriate for ordering messages or (ii) to delay them, which is appropriate for view changes so that a safe state is reached. This choice is application dependent. For example, in the passive replication style, view changes are imperative only when the coordinator crashes. By using

this approach, one avoids frequent or unsuitable changes. The communication within the group is finally completely uncoupled from the dynamic management of its group membership. The benefits are: effectiveness, flexibility and adaptability. The EDEN system follows this approach of long-lived group. To our knowledge, besides EDEN , only the family of systems developed in Lausanne (Phoenix, Bast, OGS) presents such a characteristic.

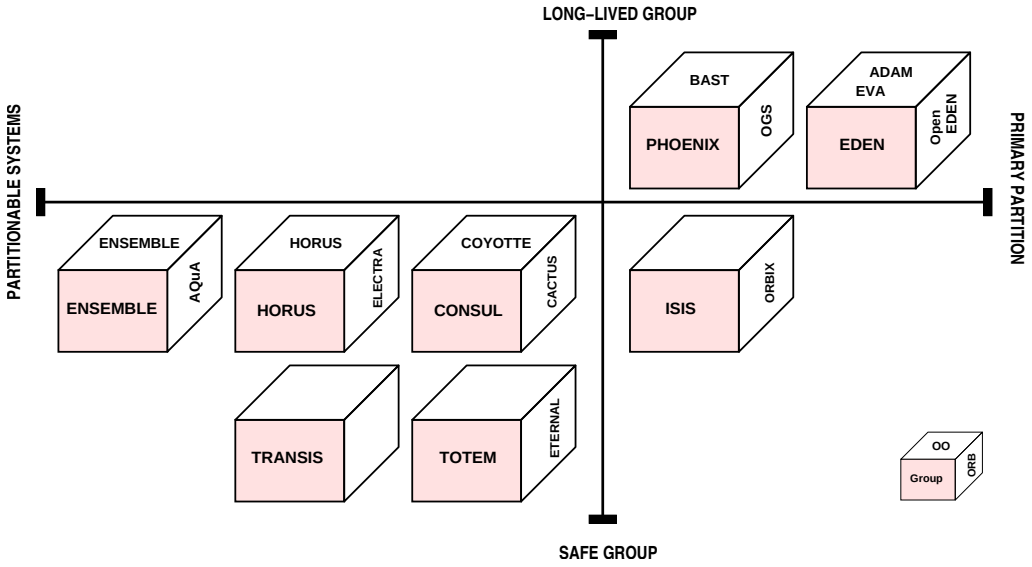

Figure 1: The group systems by type of evolutionary approach and group membership partition

Figure 1 classifies the families of group services according to the type of evolutionary approach (safe group or long-lived) and the type of group partition (primary or partitionable). In the figures which follow (this one included), the families are represented by cubes. Each face of the cube shows one of the elements of the family. The face group indicates the element which represents the core protocols, face OO indicates the element representing the object design criteria and face ORB represents the platform resulting from the integration of the group service in a distributed object architecture.

4.2. How to Structure the Protocols

We showed along this paper in what the theoretical solutions founded on the consensus support the construction of well structured and independent protocols. This approach also allows us to identify important abstractions (consensus, failure detectors, reliable broadcast, atomic broadcast, etc.) from which other reliable abstractions can be created (e.g. a replication service). Our interest now is to benefit from the modular nature of these solutions to build systems which are reusable, flexible, extensible and adaptable.

Protocol Classes. The object-oriented design and its advantages were appropriately used in the construction of Horus, Coyotte and Bast. Table 1 shows a summary of important principles introduced by these systems in the design of flexible reliable abstractions. In all these systems, the flexibility is reached by considering the “elementary reliable abstractions” as *protocol classes* from which other abstractions can be built.

Protocol Patterns. Horus and Coyotte primarily use the mechanism of inheritance to carry out protocol compositions. Bast goes further by introducing the concept of *protocol patterns* which are

domain-specific design patterns that describe how to compose protocol objects. Bast synthesizes much knowledge already established in the field of distributed systems and it identifies some important protocol patterns which are recurring in the modeling of fault tolerant applications. The `active replication` pattern and the `distributed agreement` pattern are good examples.

	Abstractions		
Group System	*Protocol Classes*	*Protocol Patterns*	*Protocol Components*
Horus	•		
Coyotte	•		
Bast	•	•	
Eden/Adam	•	•	•

Table 1: Strategies to structure protocols

Composable Protocol Stack. Horus introduces the concept of *composable protocol stack*, where each specific functionality (atomic broadcast, reliable broadcast, group membership, etc.) is implemented by a micro-protocol which can be placed above another to form a stack. This establishes a layer-based architecture in which each layer uses the services of the adjacent lower layer to provide an extended service to the adjacent upper layer, up to the last layer which provides the functionality required. The composition of the stack can be done at run time in a variety of ways to meet the exact application requirements. This way of "composition" has further been adopted by the great majority of group toolkits (Bast, Coyotte, OGS, etc.). Figure 2 shows the representation of such a strategy to implement a consensus-based replicated service.

Figure 2: Composable protocol stack

Layer Based Interaction. The Horus platform adopts a communication pattern in which an outgoing message goes through all the layers in a top-down manner, whereas an incoming message enters through the bottom layer and has to be pushed through the layers in a bottom-up way. Information is passed to lower layers using procedure calls, whereas information from lower layers is delivered to higher layers using call-backs which are installed dynamically by the higher layers when they start their execution. The same approach can be found in Bast, OGS and many other toolkits.

Event Based Interaction. Instead of using a linear pattern of communication, Coyotte represents the interactions between the protocol classes into a graph. In Coyotte the communication

between each micro-protocol is done in a dynamic way by the emission and the reception of events.

4.3. Component Oriented Design

As a layer-based architecture is frequently employed to specify communication protocols (OSI/ISO, TCP/IP, etc.), it could seem natural to keep on designing reliable protocols in the same way. However, if on one hand the layer-based design defines a simple discipline of development, on the other hand the need to accommodate the protocols functionality into layers that can only communicate with their adjacent layers imposes a restrictive model. Rather than being specified as a collection of well defined functionality layers, micro-protocols are normally presented as a description of the cooperating entities that together implement the protocol's functionality, and the interactions among them [29].

From a performance viewpoint, the layer-based style of communication yields penalties due (arbitrary delays can be introduced, the growth of the message size). This happens because information must cross all the stack to arrive at the destination. Of course, ad hoc optimizations (as for example, short cuts between non-consecutive layers) can allow alternative interaction paths and also enable a reduction on the protocol's latency [30]. Nevertheless, even with such improvements, a static stack of protocols is still not flexible enough.

To avoid the drawbacks of a layer-based architecture while keeping the advantages associated to the decomposition of the problem into well defined and independent basic services, we advocate the use of a *component oriented design*. In such an approach, the reliable service is designed as several independent components, connected via a communication platform and cooperating by well-defined interfaces. Thus, we propose to transform the "protocol classes" into *protocol components*. Instead of using a fixed hierarchy of classes to design a particular reliable abstraction, we propose to break this hierarchy into several independent entities, each one representing a reliable abstraction materialized by a component. Communication between them is not restricted to a stack, they undertake the shape of a graph. Such a flexibility allows components to be freely combined in order to create reliable services even more complex.

With such a strategy, the relationship between micro-protocols is no more made by implicit specializations. They are explicit and can be done in an asynchronous way. The functionalities of the reliable semantics can thus be implemented by a certain number of autonomous distributed objects interacting via the production and the consumption of events. In this way, events that are produced by suppliers must be routed by an event channel middleware to their corresponding consumers. Distributed algorithms are normally designed as reactive entities to the occurrence of events (messages, exception notifications, etc.). So, an event-based pattern of interaction is a simple and elegant way to fill in the gap between the specification of the protocol and its implementation.

ADAM is a library of agreement protocols, which is built according this component oriented design. In the implementation, the hierarchy of "protocol classes" (figure 2) – common in Horus, Bast and OGS – is completely broken in order to derivate the "protocol components". The EDEN group system is thus the result of the composition of some of the ADAM components.

Figure 3 represents the group systems according to their design strategy. We classify them in *composable protocols* (those designed by following OO concepts) and *non composable protocols*. Among composable protocols, we distinguish those which interact by layers and those which interact by events. Let us notice that, as in Coyotte, we propose to structure the protocols as a graph of entities. However, differently from it, we provide a library of components (ADAM) ready to be used by protocol developers in the implementation of reliable services. To our knowledge, Coy-

otte does not provide such a mechanism. As in BAST, we base ourselves on the `distributed agreement` pattern to capture the recurring structure of various classical agreement protocols.

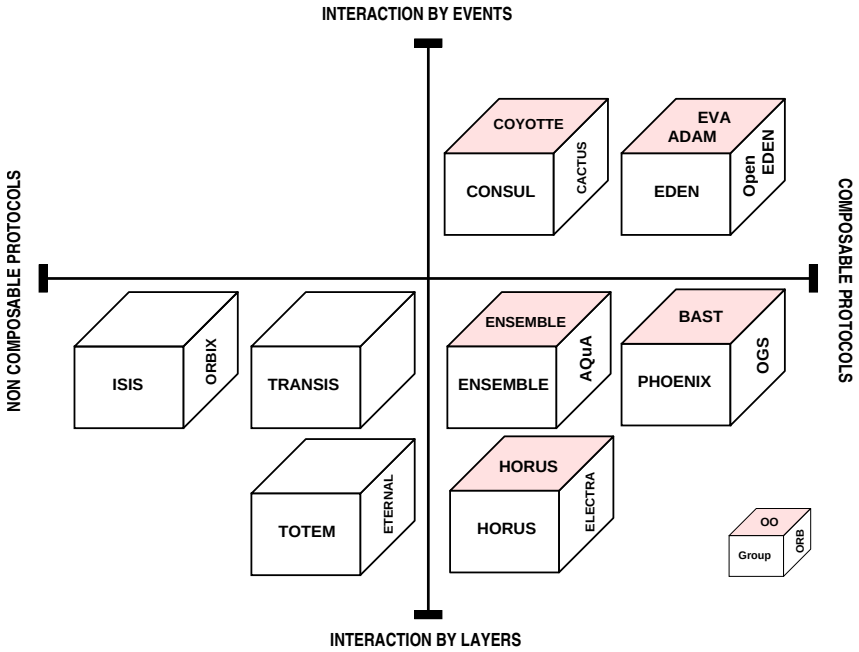

Figure 3: Group services and their design strategies

5. The EDEN Group Service

EDEN is a configurable group service based on a library of agreement components, called ADAM. ADAM [13, 14] is implemented above EVA [15], an event-based framework for developing distributed abstractions and high-level communication protocols.

ADAM is a component-based library of agreement abstractions, used to build reliable programming toolkits. The central element of the ADAM library is the GENERIC AGREEMENT COMPONENT (GAC) [29]. It implements a generic and adaptative fault-tolerant consensus algorithm that can be customized to cope with the characteristics of the environment. Moreover, thanks to a set of versatile methods, its behavior can be tuned to fit the exact needs of a specific agreement problem. A range of fundamental ADAM components are implemented as specializations of this GAC component. By composing some of the ADAM agreement components, we have built a group communication logic. An ACTIVE REPLICATION service has been designed and forms the heart of the EDEN system. It is mainly based on the ATOMIC BROADCAST, GROUP MEMBERSHIP and GENERIC AGREEMENT components. Figure 4 shows the EDEN components and their relationships.

The framework EVA implements a publish-subscriber communication environment to structure entities composing high-level protocols. In this architecture, protocols are regarded as a number of cooperating objects (entities) that communicate via an event-channel. Applications built on top of EVA are composed of a set of cooperating components that communicate with each other via the production and consumption of special types of objects called events. Further, a particular component is itself formed by cooperating entities (*e.g.* passive and active objects, sub-components, etc.) which also communicate via a similar event channel mechanism. Each component has an event channel manager that is responsible for routing the events produced by

its supplier entities to all of its consumer entities that have register to receive a notification for the particular event.

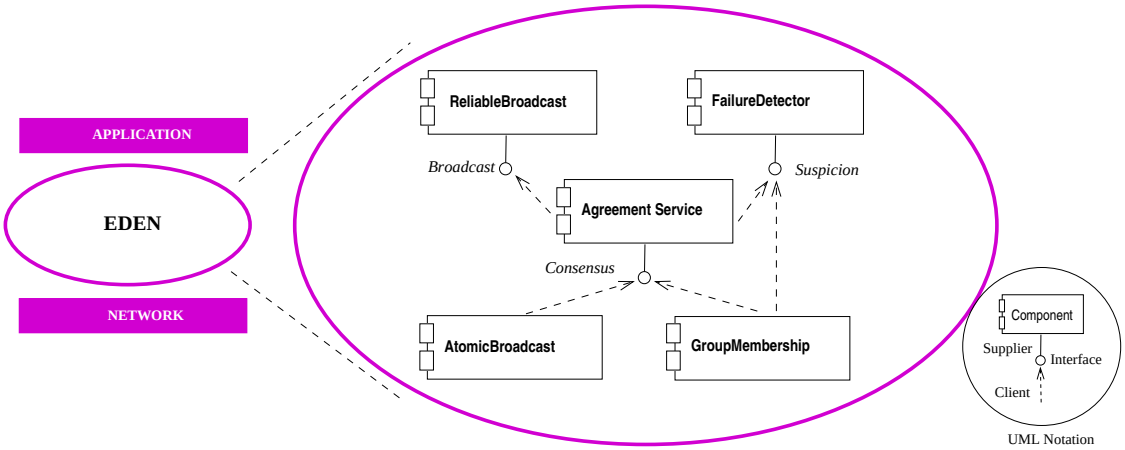

Figure 4: The EDEN Components

The ADAM library currently includes the most important components for a reliable distributed programming. Some of them are:

- GENERIC AGREEMENT COMPONENT (GAC). It lies at the core of the ADAM library. It implements a generic fault tolerant consensus algorithm [29]. To be operational, it has to be instantiated through the definition of a concrete agreement component (e.g. group membership, atomic broadcast). Indeed, it was designed to solve multiple agreement problems at the same time.
- GROUP MEMBERSHIP MANAGEMENT (GM) It is in charge of managing the changes on the group composition [31]. It will be composed with GAC in order to get the agreement on three sets of objects: (i) those which are authorized to *join* the group (ii) those which are authorized to *leave* the group (iii) those which are suspected to be faulty. The GM component interacts with a FAILURE DETECTOR component to take into consideration crashed members.
- ATOMIC BROADCAST (AC) It is implemented by a set of AC components that are in charge of receiving the requests broadcasted by external clients and to deliver them in the same order. For this it will interact and make use of the service provided by the GAC component.
- VIEW SYNCHRONY To install a new view, all the non-crashed members of the previous view must have delivered all the messages ordered during the previous view. The VIEW SYNCHRONY is in charge of ensuring this property. To achieve this goal, no additional agreement is necessary. Indeed, strong synchronizations have to be implemented to ensure that ATOMIC BROADCAST and GROUP MEMBERSHIP observe their decisions and progress in a consistent way. In practice, the installation of the new view is postponed till all the synchronization constraints are satisfied.

6. Conclusion

This paper was dedicated to present the design issues of the EDEN group system. It was conceived by following two innovative approaches which contribute to its flexible and adaptable character. From an algorithmic point of view, it is based in theoretical results that precisely characterizes the liveness and safety properties ensured by the protocols. This allows better control of the

behavior of the applications in the occurrence of dysfunctions. Moreover, it advocates the use of a component oriented design to implement each elementary reliable abstraction (atomic broadcast, group membership, etc.). In such an approach, a reliable application will be designed as the composition of several independent components, connected via a communication platform and cooperating by well-defined interfaces.

References

[1] K. Birman, *Reliable Distributed Computing with the ISIS Toolkit*, ch. Virtual Synchrony. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[2] R. van Renesse, K. Birman, and S. Maffeis, "Horus: a flexible group communication system," *Communications of the ACM*, vol. 39, pp. 76–83, Apr. 1996.

[3] M. Hayden, *The Ensemble System*. PhD thesis, Cornell University, 1998.

[4] K. P. Birman, R. van Renesse, and W. Vogels, "Spinglass: Secure and scalable communications tools for mission-critical computing," in *International Survivability Conference and Exposition. DARPA DISCEX-2001*, (Anaheim, California), June 2001.

[5] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg, "On the formal specification of group membership services," Tech. Rep. TR95-1534, Depto of Computer Science, Cornell University, Aug. 1995.

[6] G. V. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: a comprehensive study," *ACM Computing Surveys*, vol. 33, pp. 427–469, Dec. 2001.

[7] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of ACM*, vol. 32, pp. 374–382, Apr. 1985.

[8] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost, "On the impossibility of group membership," in *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, (New York, USA), pp. 322–330, ACM, May 1996.

[9] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of ACM*, vol. 43, pp. 225–267, Mar. 1996.

[10] R. Guerraoui and A. Schiper, "The generic consensus service," *IEEE Transactions on Software Engineering*, vol. 27, pp. 29–41, Jan. 2001.

[11] B. Garbinato, *Protocol Objects & Patterns for Structuring Reliable Distributed Systems*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998.

[12] F. G. P. Greve, *Réponses efficaces au besoin d'accord dans un groupe*. PhD thesis, IRISA - Université de Rennes I, France, Nov. 2002.

[13] F. Greve, , M. Hurfin, J.-P. L. Narzul, M. Xiaojung, and F. Tronel, "A library of agreement components for reliable distributed programming," in *Workshop on Communication Abstractions for Distributed Systems, with ACM ECOOP - 17th European Conference on Object-Oriented Programming*, (Darmstadt, Germany), 2003.

[14] F. Greve, , M. Hurfin, and J.-P. L. Narzul, "Adam: une bibliothèque de composants d'accord pour la programmation d'applications fiables," in *Actes des Journées Composants*, (Lille, France), Mar. 2004.

[15] F. Brasileiro, F. Greve, M. Hurfin, J.-P. L. Narzul, and F. Tronel, "Eva: an event based framework for developing specialised communication protocols," in *NCA 2001: IEEE International Symposium on Network Computing and Applications*, pp. 108–119, Feb. 2002.

[16] F. Cristian, "Reaching agreement on processor group membership in synchronous distributed systems," *Distributed Computing*, vol. 4, pp. 175–187, Apr. 1991.

[17] V. Hadzilacos and S. Toueg, *Distributed Systems*, ch. Fault Tolerant Broadcasts and Related Problems, pp. 97–145. Addison-Wesley, 1993.

[18] K. M. Chandy and J. Misra, "How processes learn," *Distributed Computing*, vol. 1, pp. 40–52, 1986.

[19] S. Maffeis, "Electra—making distributed programs object-oriented," in *Proc. of the Usenix Symp. on Experiences with Distributed and Multiprocessor Systems*, (San Diego, CA (USA)), pp. 143–156, 1993.

[20] Y. Ren, *AQUA: A Framework for Providing Adaptive Fault Tolerance to Distributed Applications*. PhD thesis, University of Illinois, Urbana, 2001.

[21] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos, "Totem: a fault-tolerant multicast group communication system," *Communications of the ACM*, vol. 39, pp. 54–63, Apr. 1996.

[22] D. Dolev and D. Malki, "The transis approach to high availability cluster communication," *Communications of the ACM*, vol. 39, pp. 64–70, Apr. 1996.

[23] C. Malloth, *Conception and Implementation of a Toolkit for Building Fault-Tolerant Distributed Applications in Large Scale Networks*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1996.

[24] P. Felber, *The CORBA Object Group Service: A Service Approach to Object Groups in CORBA*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998.

[25] S. Mishra, L. Peterson, and R. Schlichting, "Consul: a communication substrate for fault-tolerant distributed programs," *Distributed Systems Engineering Journal*, vol. 1, no. 2, pp. 87–103, 1993.

[26] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu, "Coyote a system for constructing fine-grain configurable communication services," *ACM Transactions on Computer Systems*, vol. 16, Nov. 1998.

[27] M. A. Hiltunen and R. D. Schlichting, "The cactus approach to building configurable middleware services," in *Proc. of the Workshop on Dependable System Middleware and Group Communication (DSMGC 2000)*, Oct. 2000.

[28] F. Greve, , M. Hurfin, J.-P. L. Narzul, M. Xiaojung, and F. Tronel, "A library of agreement components for reliable distributed programming," in *Workshop on Communication Abstractions for Distributed Systems, with ACM ECOOP - 17th European Conference on Object-Oriented Programming*, (Darmstadt, Germany), 2003.

[29] M. Hurfin, R. Macêdo, M. Raynal, and F. Tronel, "A generic framework to solve agreement problems," in *Proc. of the* 19^{th} *IEEE Symposium on Reliable Distributed Systems (SRDS'99)*, (Lausanne, Switzerland), pp. 56–65, Oct. 1999.

[30] X. Liu, C. Kreitz, R. van Renesse, J. Hickey, M. Hayden, K. Birman, and R. Constable, "Building reliable, high-performance communication systems from components," in *Proc. of the* 17^{th} *ACM Symposium on Operating Systems Principles (SOSP'99)*, (Charleston, USA), pp. 80–92, Dec. 1999.

[31] F. Greve, M. Hurfin, M. Raynal, and F. Tronel, "Primary component asynchronous group membership as an instance of a generic agreement framework," in *ISADS'2001: 5th International Symposium on Autonomous Decentralized Systems*, pp. 93–100, Mar. 2001.