# Communication Abstractions for Distributed Systems*

Antoine Beugnard[1], Ludger Fiege[2], Robert Filman[3], Eric Jul[4], Salah Sadou[5], and Eiko Yoneki[6]

[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
[6] University of Cambridge, UK
eiko.yoneki@cl.cam.ac.uk

**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

Applications have become increasingly distributed. Distribution complicates systems building and exacerbates problems such as dealing with failure, and providing security, quality of service, reliability, and manageability.

System development is eased by abstraction and modeling. How should we model distributed systems? Distributed systems can be understood as communicating objects. To tackle the problems of building distributed systems, it is useful to focus on the abstract issues of inter-component communication. Examples of distributed communication mechanisms include messaging systems, remote procedure calls, distributed objects, peer-to-peer and publish-and-subscribe. Within any such paradigm, there are many opportunities for specialized and detailed engineering decisions. While mechanisms such as these are a good foundation for

* The title of this report should be referenced as "Report from the ECOOP 2004 Workshop on Communication Abstractions for Distributed Systems".

J. Malenfant and B.M. Østvold (Eds.): ECOOP 2004, LNCS 3344, pp. 67–75, 2004.

dealing with the problems of distribution, there remain many issues about how to mold these ideas to deal with the problems of real systems.

At the previous ECOOP workshops, we identified some problems (security, privacy, partial failure, guaranteeing quality of service, run-time evolution, meta-object protocols, and ordering of events) that are important concerns of any communication abstraction. The goal of this workshop was to contrast and compare communication abstractions for distributed systems. Participants were asked to submit a position paper on some aspect of communication abstractions for distributed systems. To focus the groups discussion, this year we considered the distributed aerospace information problem, described in the call-for-papers. Prospective participants were requested to relate their contribution to some facet of that that problem. The workshop itself consisted of short presentations, discussion of those presentations, and division into smaller topic study groups.

We received 8 positions papers. All were reviewed by at least two members of the program committee and 6 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 two 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. This year, the chosen topics were "Events" and "Dealing with Errors"

## 2 Position Papers Abstracts and Discussions

### 2.1 Communication Abstraction and Verification in Distributed Scenario Integration [1]

Paper written by Aziz Salah, Rabeb Mizouni and Rachida Dssouli and presented by A. Salah from the department of C.S. University of Quebec, Montreal (Email: salah.aziz@uqam.ca)

Successfully modeling and analysing requirements are among the main challenges to face up when it is time to produce a formal specification for distributed systems. Scenario approaches are proposed as an alternative to make this process easier. They are based on the decomposition of the system behaviour into intuitive pieces that are distributed scenarios. In this paper, we propose an approach to detect the unspecified reception errors by the integration of scenarios. The decision about such property is made possible according to an architectural communication model, which states the communication abstraction level we are considering. We synthesize from message sequence char a set of automata, one per object. Then, we give decision procedure for unspecified reception faults.

### 2.2 Realizing Large Scale Distributed Event Style Interactions [2]

Paper written by A Vijay Srinivas, Raghavendra Koti, A Uday Kumar and D Janakiram and presented by D Janakiram from the Distributed & Object Systems Lab, Dept. of Computer Science & Engg., Indian Institute of Technology, Madras, India (Email: fdjram@cs.iitm.ernet.in).

Interactions in distributed object middleware that are based on Remote Procedure Call (RPC) or Remote Method Invocation (RMI) are fundamentally synchronous in nature. However, asynchronous interactions are better suited for large scale distributed systems. This paper presents the design and implementation of an asynchronous event based communication paradigm. Typed events, fully distributed hierarchical event dispatching as well as causal delivery of events are the key features that are supported. The paradigm has been realized over Virat, a wide area shared object space that we have built. Virat uses replication, caching and distributed services as the main concepts and provides a well published interface, as well as various relaxed consistency models. This communication abstraction is especially beneficial to applications such as modernizing Airspace Systems, where scalable asynchronous event notification is a critical issue.

### 2.3 Event Brokering Over Distributed Peer-to-Peer Environments [3]

Paper written and presented by Eiko Yoneki from University of Cambridge Computer Laboratory Cambridge, United Kingdom (Email: eiko.yonekig@cl.cam.ac.uk).

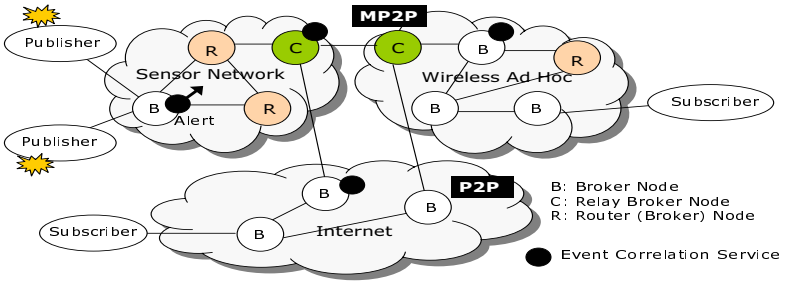


**Fig. 1.** Event Broker Grids over P2P and MP2P Networks

Peer-to-peer (P2P) networks offer a promising paradigm for developing efficient distributed systems. Event-based middleware (EBM) is becoming a core architectural element in such systems. EBM is based on publish/subscribe communication paradigms that became popular providing asynchronous and multipoint communication. Most distributed EBM contains three main elements: a producer who publishes events, a consumer who subscribes his interests to the system, and an event broker with responsibility to deliver the matching events to the corresponding consumers. The first event-based middleware systems were

based on the concept of channel or topic communication. In an attempt to overcome the limitation on subscription declarations, the content-based model has been introduced, which allows subscribers to use flexible querying languages to declare their interests with respect to event contents.

There are diverse network environments from Internet-scale P2P systems to sensor networks. Event broker grids need to communicate over wired P2P networks, wireless ad hoc networks (WAHN) or even Web services. In WAHN, the combination of mobile devices and ad-hoc networks are best managed by the creation of highly dynamic, self-organizing, mobile P2P systems (MP2P). Mobile hosts continuously change their physical location and establish peering relationships with each other based on proximity. Asynchronous communication is essential to support anonymous coordination of communication in such ubiquitous environments. We previously introduced publish/subscribe semantics in WAHN [4]. Note that when publish/subscribe becomes powerful in WAHN environments, all the nodes may not be in pure ad hoc topology, and some nodes must be connected to the Internet backbone or relay nodes from different network environments. For example, a broker node can act as a gateway from sensor networks operating data aggregation and distributes them to the other mobile networks based on its contents (see Fig. 1). Broker nodes that offer event correlation services can coordinate data flow efficiently. Thus, the way publish/subscribe systems can be constructed based on the data contents, which triggers related chained actions. Unusual events will be detected by the embedded sensors triggering distribution of subsequent events to the entire system.

This paper presents a vision of an event brokering system over mixed P2P networks in a multi event broker model. In addition, an event condition action engine will be integrated that is required to perform event correlation services within brokers. Event correlation service combines the information collected by individual resources into higher level information or knowledge. Events reflect the movement and flow of information. Combining event-condition-action-rules, complex event composition, data aggregation, and detection with event-based systems will provide a way to construct large distributed systems providing abstraction of communication over mixed network environments.

### 2.4 Homogenization: A Mechanism for Distributed Processing Across a Local Area Network [5]

Paper written by Mahmud Shahriar Hossain, M. Muztaba Fuad, Debzani Deb, Kazi Muhammad Najmul Hasan Khan, and Dr. Md. Mahbubul Alam Joarder. (Email: shahriar-cse@sust.edu)

Distributed processing across a networked environment suffers from unpredictable behavior of speedup due to heterogeneous nature of the hardware and software in the remote machines. It is challenging to get a better performance from a distributed system by distributing task in an intelligent manner such that the heterogeneous nature of the system do not have any effect on the speedup ratio. This paper introduces homogenization, a technique that distributes and balances the workload in such a manner that the user gets the highest speedup

possible from a distributed environment. Along with providing better performance, homogenization is totally transparent to the user and requires no interaction with the system.

### 2.5 Modelling a Distributed Network Security Systems Using Multi-agents Systems Engineering [6]

Paper written by Gustavo A. Santana Torrellas from Instituto Mexicano del Petrleo Mantenimiento y Perforacin de Pozos (Email: gasantan@imp.mx)

Recent developments have made it possible to interoperate complex business applications at much lower costs. Application interoperation, along with business process reengineering can result in significant savings by eliminating work created by disconnected business processes due to isolated business applications. However, we believe much greater productivity benefits can be achieved by facilitating timely decision-making, utilizing information from multiple entreprise perspectives. To stay competitive in this current scenario, it is crucial for organizations to react quickly to changing security factors, such as virus attack, active intrusion, new technologies, and cost of disaster recovery. Such information security changes often encourage the creation of new security schemas or security improvements. Accommodating frequent systems information changes requires a network security system be more flexible than currently prevalent systems. Consequently, there has recently been an increasing interest in flexible network security and disaster recovery systems.

### 2.6 But What If Things Go Wrong? [7]

Paper written and presented by Johan Fabry from Vrije Universiteit Brussel (Email: Johan.Fabry@vub.ac.be)

Nowadays, building distributed systems is said to be easy: just use one of the many distribution frameworks out there, and all the hard stuff will be taken care of for you. However, when we focus on what to do when things go wrong, i.e. consider partial failure, we see that examples in the literature will either ignore these kinds of exceptions, or stop the program, which is clearly inadequate.

A generic and useful failure-handling mechanism for distributed systems exists in the from of transactions, however a significant issue with transactions is that they may be rolled back by the transaction manager to break deadlocks. Again, looking at the literature, we see no thorough treatment of this extra kind of failure, except for the proposal of a number of Advanced Transaction Mechanisms (ATM) that handle this. A large number of ATMS can be found in the literature, and two books have been published about the subject. So, while ATMS exist to solve these problem, we still see no use of them in commercial systems, even although this is research from the 80s and 90s. Indeed, we cannot even find the most well-known model: nested transactions, commercially.

It is our position that the problem with ATMS lies in the difficulty for the application programmer to specify how to use these mechanisms. Given their nature, an ATMS needs more information about the transaction than a classical

system. We think that, since letting the developer specify error-handling code is already troublesome, going the extra mile to use a ATMS is nigh-on impossible.

Therefore, we should support the application programmer when specifying these advanced transactions. The programmer should reason about this extra information at a higher level of abstraction, and can specify the required transactional properties at this higher level, e.g. by using a Domain Specific Language.

We have implemented a first prototype tool, outlined at the workshop, as a first step toward this goal. Notable functionality is the ability to provide generic deadlock- or exception-handling strategies for a transaction such as simply retrying the transaction. The question to attendees was what generic deadlock- or exception-handling strategies that are worthwhile to offer the programmer.

**Questions and Answers.** A first question was how our generic error-handling strategies interact with possible recovery work already done by the transaction monitor. The answer is that we do not really see any interaction, we consider our work more as a 'second line of defense' for when the transaction monitor fails to recover.

The remainder of the discussion was more related to the depth of generalized error-handling strategies: we can go very deep, e.g. do investigations of the program history, and apply AI techniques, which is done by NASA. We do not aim to cover such complex situations, and imagine everything that can possibly go wrong. Instead we are looking for general and simple ways in which things get fixed when something goes wrong, and providing programmer abstractions for these fixes.

## 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 two of them for further exploration: *dealing with errors* and *event based communication in distributed systems*.

### 3.1 Dealing with Errors

We started with the point that usually, specifications focus on what should happen and forget, sometimes voluntarily, the description of what to do when things go wrong! May be because the former is a single description (that obviously can be complex in itself) while the latter is very, very open.

In an attempt of modelling what is needed to deal with errors we identified four tasks:

1. Monitoring. It is essential to be able to observe what happens in the system. Where is this code? Complex architectures are layered, and monitoring code is probably scattered. Error management has to be taken into account from the lowest levels to the application level.

2. Detecting. From the low-level monitoring data, the system must infer bad risky situations. Perhaps a dedicated language is needed to define theses unexpected events. A more elaborate error mechanism may also be able to express the level of risk and urgency of reaction required for particular situations.
3. Repairing actions. A set of "atomic" corrective actions is needed. For instance: restart, do something & restart, do something & continue, abort, do something & abort, abort & do something. Any real system will have to deal with the problem that these actions are themselves subject to errors.
4. Managing. It's easy to imagine situations where not just a single alarm is sounded. Overall, alarms and repairs may occur simultaneously, and rules and mechanisms are needed to arbitrate among them. Rules are required to set up a hierarchy of decisions when things go worst and worst ...

This view of the world relies on the existence of stable system states (recovery checkpoints) that need to be defined and implemented. However, many errors are not recoverable. This is especially the case for error situations after interactions with the external world — one can not unprint something from the printer or convince the user that he or she didn't really see something on the screen. The procedure to correct or undo wrong or bad real-world side-effects is often out of the scope of the software system. It may require legal or social actions when possible, and sometimes, is completely impossible — all the kings horses and men can reassemble neither Humpty Dumpty nor an Ariane V rocket.

### 3.2 Event-Based Communication in Distributed Systems

This workshop has always included papers whose communication abstract was based on "events."

Distributed applications usually exist in heterogeneous environments (systems and/or languages). Middleware serves to mediate communications. Its goal is to allow various systems and/or languages to share the same communication protocol. The complexity of the implementation and the use of this common protocol depend on the choice of the communication mode. In this working group, we made a comparison between two modes of communication: event-based and method-based. In object oriented systems, the communication may require the following element:

**Interface:** the type of the recipient of the message;
**Data Type:** the signature of the called method or the type of forwarded information;
**Identity:** recipient's address.

The following table shows the needs for each of the two selected modes:

| Comm. mode | Interface | Data type | Identity |
|---|---|---|---|
| Event-based | No | Yes | No |
| Method-Based | Yes | Yes | Yes |

We notice that the communication based on event is less demanding. This implies less dependency between sender and receiver and thus minimizes the role of the common communication protocol in a heterogeneous environment. Indeed, whereas a method-based middleware requires common modes of identification and data representation, an event-based middleware requires only a common mode of data representation. We think that in the future the event-based middleware should be generalized.

**How Should We Construct Different Communication Types Using Events?** In this section we visit the various types of communication to imagine their implementation in an event-based system:

**Asynchronism:** In contrary to method-based communication, event-based communication is naturally asynchronous. Objects send events and react to others. No bond is necessary between the sent events and the those received.

**Synchronism:** On the other hand, the event-based communication requires a mechanism to carry out an equivalent to synchronous calls. An example of such mechanism would be the acknowledgement of receipt, which is sent by the receiver at the end of its task. This defines the termination of the event sender's waiting . The acknowledgement of receipt is an event which contains an identifier provided first by the event sender.

**Transaction:** In this mode of communication, the receiver must deal with a complete succession of calls and respect its order. This mechanism may be produced by the concept of composite events. A composite event is a structured set of events (a list for example). The structure defines the transaction's order.

**Call with Return:** As in the case of synchronous calls, to carry out calls with return, it will be necessary to add an identifier to the event in order to associate it with the event representing its result.

**Broadcast:** Broadcast corresponds to the sending of public events. Any object interested by those events can react.

**Multicast/Unicast:** This mechanism requires the concept of private event. In the case of method-based calls, it is often the sender object which determines the subset of the concerned objects by its call. To approach this mechanism, we can imagine to add, to the event management system, the concept of filters. On one side the sender gives a filter with its event and on the other side, the receivers can choose filters during the subscribe to a type of events.

**How Does This Help?** The principal characteristic of the event-based systems is the decoupling between the objects participating in the communication. This characteristic facilitates the implementation of mobile computing and the peer-to-peer network environments. There is a series of workshops on the topic of distributed event-based systems and the last one is co-localized with ICSE'04 (*http://serl.cs.colorado.edu/ carzanig/debs04/*).

## 4 Workshop Conclusions

A primary goal of this workshop is to enable the sharing of concepts and technologies among various communities. It is clear to us that a critical element for this goal is creating a shared understanding and categorization of communication mechanisms. This is a long-term goal. In this year's workshop, we have made progress on clarifying the nature of event-based communication, and, more innovatively for this workshop series, began to address the issue of dealing with errors in communications. This latter issue is clearly critical to the development of real systems and argues for the development of the right abstractions to support that real system development.

All previous workshop position papers are available via the CADS wiki site *http://wiki.enstb.org/cads/*.

## References

[1] Aziz Salah, Rabeb Mizouni, and Rachida Dssouli: Communication abstraction and verification in distributed scenario integration. `http://bscw.enst-bretagne.fr/pub/bscw.cgi/0/2237034`, July 2004.

[2] A. Vijay Srinivas, Raghavendra Koti, A. Uday Kumar, and D. Janakiram: Realizing large scale distributed event style interactions. `http://bscw.enst-bretagne.fr/pub/bscw.cgi/0/2237034`, July 2004.

[3] Eiko Yoneki: Event brokering over distributed peer-to-peer environments. `http://bscw.enst-bretagne.fr/pub/bscw.cgi/0/2237034`, July 2004.

[4] E. Yoneki and J. Bacon: Content-based routing with on-demand multicast. In: Proc. 23rd ICDCS Workshop - WWAN, March 2004.

[5] Mahmud Shahriar Hossain, M. Muztaba Fuad, Debzani Deb, Kazi Muhammad Najmul Hasan Khan, and Dr. Md. Mahbubul Alam Joarder: Homogenization: A mechanism for distributed processing across a local area network. `http://bscw.enst-bretagne.fr/pub/bscw.cgi/0/2237034`, July 2004.

[6] Gustavo A. Santana Torrellas: Modelling a distributed network security systems using multi-agents systems engineering. `http://bscw.enst-bretagne.fr/pub/bscw.cgi/0/2237034`, July 2004.

[7] Johan Fabry: But what if things go wrong? `http://bscw.enst-bretagne.fr/pub/bscw.cgi/0/2237034`, July 2004.