Using Space-Based Computing for More Efficient Group Coordination and Monitoring in an Event-Based Work Management System

Marcus Mor, Richard Mordinyi and Johannes Riemer
Institute of Computer Languages, Vienna University of Technology
[mor|richard|jr]@complang.tuwien.ac.at

Abstract

Group communication is a very difficult task to be implemented in distributed applications. Particularly, work management systems are important in many industries to support the coordination of distributed groups of mobile workers with different levels of availability. Traditional event-based systems using point-to-point communication such as e-mail are not well suited to coordinate a work group as the state of a work item is not always clear and this mode of communication creates many mistakes and massive communication overhead because those tasks are solved via a central server. In this paper, we analyze a work process in a major insurance company, develop a prototype providing solutions for the problems by exploring the coordination features deployed in space-based computing and compare the current system with the prototype.

1. Introduction

Collaborative applications such as in computer supported cooperative work (CSCW), groupware, or social software essentially require a basis for well-founded coordination means. As for stationary information systems these are well established. Recent developments show that the market of mobile devices and services is rising ([23],[24]) and thus becomes more and more relevant for collaborative applications. Information systems are steadily transforming themselves into mobile information systems because of increased availability and capability of mobile and wireless technologies and portable devices at reduced cost. Due to more challenging requirements of mobile environments, more appropriate coordination mechanisms are necessary [3]. Such environments are characterized by a high degree of *error proneness*, *steady disconnections*, *frequently changing topologies* and *conditions*, *lower transmission rates* and *resource-restricted devices*. Hence, system designers of architectures for mobile applications have to provide flexibility to a great extent in order to abstract from these unfavorable conditions.

We claim that the inherent properties of the peer-to-peer (P2P) paradigm seem to be highly applicable to mobile environments on the one hand and to address the increased claim for coordination in this context on the other. The essential properties are ([1],[2]): the decentralized nature (*no single point of failure*), *fault-tolerance* and *robustness* through *redundancy* and *replication*, *scalability* and *adaptability* (changes in the environment are masked by the overlay network), *loose couplings* through *time* and *location transparency*, autonomy of the nodes, and dynamic assignment of roles, which accumulates to the required flexibility property.

In this paper, we introduce a business case (chapter 2) that will be the basis for the case study. Its current situation is described and coordination related requirements for an improved solution are identified. We investigate four different technology approaches in chapter 3 to find out to which extend they help to realize those requirements. In chapter 4, we reason about the selected coordination technology for the prototype of an improved solution and detail the implemented prototype. We give a comparison between the technology used by the business case and the new prototype in chapter 5 and end the paper with the conclusion about gained experiences in chapter 6.

2. Case description

The business case we are using for the case study was taken from a major insurance company, where many agents are set in field services. The decentralized workers, equipped with mobile devices, are working on their own visiting costumers (e.g. a car driver after an accident controlling the claim). Therefore, they use a "Prisma Client" [25] – like a mail client – to collect their data (e.g. costumer data, reports on claims, …) in local databases (see Figure 1). The data (e.g. new tasks) they receive from the "Prisma Server", a mail server that provides an inbox and outbox for each user, are also put there.

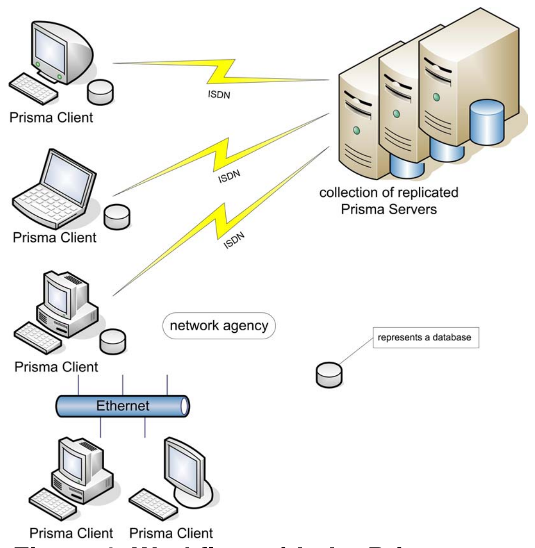

Figure 1. Workflow with the Prisma system

An example for the application of this system: Insurance policies are sold via a software application. Every new policy is stored in the local database of the client first. After the stored data is complete, they are moved into the outgoing mailbox. The network connection has to be built up, and the transfer process has to be started manually. The data are then transferred via FTP to the host (providing the Prisma Server). After the transmission to the server, new data in the inbox for the user will be sent back. After the transfer of data is complete, client and server will start some automatic or semi-automatic (user interaction being necessary) tasks that will process the data locally.

Each client (and user) transmits its work tasks independently to the central server. The supervisors are not notified about those happenings (e.g. they are not informed that a new contract was finished).

Current *event-based systems using point-to-point communication* such as an e-mail system are not well suited to coordinate distributed work groups, as the state of work items is not always clear. This complicated and actually old type of peer-to-peer event message passing communication system between the different actors in that circle of processing asks for a better solution. The following requirements have been identified:

Monitoring: Supervisors cannot find out about the work progress of their agents. They are informed after the work has been done and do not know who received which tasks at what time. To get rid of this process the following change was introduced at some agencies. All transmissions out of the agents have to go through the supervisors' mailbox. This creates an obvious bottleneck at a supervisor. This solution gives the supervisor the information needed but produces new work as the supervisor has to transfer the data manually. An improved system should enable flexible monitoring without sacrificing efficiency.

Load balancing: The current system does not allow the supervisors to have any influence on the distribution of work packages. Packages may only be assigned to a single agent, not to a group of agents having the same skills. More flexible and dynamic load balancing is required.

Mobile client support: Another task is to find solutions to handle the mobile scenario, as the mobile infrastructure (wlan, umts) are getting more common nowadays creating the problems sketched in the introduction. To support offline clients, replication of relevant data is necessary. A problem known as "synchronizing to hell" means that most of the data is synchronized and copies are made between the clients no matter whether they are really needed. Instead, efficient, selective replication is necessary.

Performance and fault tolerance: The current solution is built on the client/server network architecture that allows many clients to connect to one single server. This creates two problems that need to be overcome: a bottleneck – all clients have to communicate with this server and a single point of failure – as there is no alternative. More powerful technologies are needed to solve these drawbacks.

3. Coordination theory

Usually distributed system applications consist of a number of components that might be either processes or software components [10]. These components are often distributed which means that the services they offer run at different computers at different locations. In order to fulfill the given task, the application has to coordinate the various components using communication facilities.

According to [7] and [13] coordination models define abstract frameworks for modeling the composition of interacting components and are mainly defined by three elements: (1) the *coordination entities* – either physical or logical – which have to be coordinated. These can be data (structures), software processes, services, agents, or even human beings interacting with computer-based systems. (2) The *coordination medium* representing the abstraction and serving as connectors between the entities and enabling the interaction among those, which is a mandatory prerequisite for direct coordination ([26], [20]) and (3) the *coordination laws*, specifying what kind of coordination activities can be performed on the medium by the entities.

Computer networks and their protocols do not provide the necessary technology for coordination facilities, therefore four major communication paradigms are analyzed in the following sections.

3.1. Message passing

The message passing (MP) paradigm means that components communicate through the explicit sending and receiving of messages (point-to-point like in a mail system). Messages are sent by the client part and received by the server part of the interaction. Messages contain specific information of the task to be carried out by the message receiver.

The development of an application using message passing seems to be easy since the communication protocol is the only barrier that has to be agreed on. The component has to know and explicitly name its communication partner (spatial coupling), and for successful communication both components have to be up and running at the same time (temporal coupling).

An early approach to realize communication in a distributed system based on message passing was to establish direct communication paths with sockets, RPC or RMI. With this kind of communication, each component interacts exactly with those components that can offer the requested services. In the worst case, each component communicates with every other component in the system resulting in a fully meshed topology. Working with RPC or RMI may have additional drawbacks concerning distributed application programming [9]. Building simple client/server applications is not too time consuming. However, when it comes to highly distributed systems, the MP approach has significant drawbacks. Due to its inherent characteristics such as spatial and temporal coupling, important topics in distributed systems programming are difficult to address. Other issues whose realization is complex with the MP systems are minimal latency, concurrency, memory access, partial failure, or scalability, and therefore design and implementation efforts are usually relatively high [10].

Publish / subscribe communication decouples message senders (publishers) from message receivers (subscribers). The system routes messages based on topics or message contents. In the first place, these systems were meant for distribution of information only. Bidirectional communication had to be emulated leading to problems such as scalability.

3.2. Service-oriented architecture

Service-oriented architectures (SOA) deal with the issue of designing and building systems by using heterogeneous, network addressable software components. Looking at the historical evolution, the term “service” has been used in many different architectures reaching from transaction monitors in the early 1990s to today’s client/server architectures and web service architectures. Following the evolution process, service-oriented architectures have reached an evolution stage where the basic concepts have been widely accepted. Typically, a service-oriented architecture consists of the following main concepts [16]: *service components* (encapsulate a specified functionality), *contracts* (describe the interfaces to the service), *containers* (represent the software execution environment), *connectors* (are responsible for the message transport and thus for inter-operability), and *discovery* (comprise mechanisms to announce, search, find, and deploy services; typically implemented as registries such as yellow pages).

The Jini technology [12] is a representative of SOA. The main components are a service provider, a service consumer and a lookup service (i.e. service locator and registry) constituting the coordination entity of the coordination model. Although the Jini specification is fairly independent with respect to the communication protocol, the coordination medium is usually conducted via Remote Method Invocation (RMI) based on TCP/IP in reference implementations. Thus, very weak time decoupling mechanisms are an unfavorable consequence. Coordination laws are addressed via a central coordination entity - the lookup service, which is responsible for the service mediation. The deployment of the service is conducted in a P2P manner between the service provider and consumer. Hence, the only pattern occurring in a Jini P2P system is the matchmaker variant of the broker pattern [30].

3.3. Multi-agent systems

An agent is a problem solving entity focused on a specific task and embedded in an environment – the agency – which provides all the necessary functionality for the agent to exist and to co-operate with other agents in order to fulfill their design objectives [21]. These agents are *autonomous* (control of their actions and behavior to a certain extent), *proactive* (capability to act in a goal directed manner, not just react to external stimuli), and *possess social abilities* (communication and negotiation with other agents in order to achieve their overall goal). An enhancement to this agent concept is the BDI approach [14] where agents are modeled comprising their individual believes, desires and intentions (BDI) which trigger their actions.

Java Agent Development Framework (JADE) complies with the BDI idea and the FIPA (Foundation for Intelligent Physical Agents) specification [27] for interoperable, intelligent multi-agent systems (MAS). Apparently, the agents represent the coordination entities. Even the agency - the so-called agent management service - is designed as a decentralized management entity consisting of a magnitude of agents distributed to each participant [28]. Communication between agents is based on message passing leading to an abolishment of any temporal constraints (temporal decoupling). Moreover, JADE also provides the possibility of intra-platform agent mobility (code location transparency [11]). Furthermore, no coordination pattern is inherently predefined in MAS. Most of the patterns, however, seem to be realizable with minor efforts due to the support of ontology definitions. By nature, the negotiation pattern is highly applicable to multi-agent systems.

3.4. Space-based computing

In space-based computing (SBC) components of the distributed application use a space for communication. The focus is on the data itself that may be transferred between the components. The notion of a message is not important any more [29]. So called shared data objects are used for the communication that might also be known under the term “blackboard-based communication model” [4].

This leads to some advantages compared to the message passing paradigm, which results in a very flexible system design. First of all the participants do not know anything about each other. This makes it possible to exchange data connectionless and anonymously since the blackboard is used to store and retrieve messages. The components do not need to share the same process or machine, but most important the participants are *temporally decoupled.* Additionally, blackboard models also provide more security because any execution environment can fully monitor and log all the interactions that occur through its local blackboard.

One of the most popular representatives for space-based computing is the “Linda-like tuple space”. It extends the blackboard model by organizing data in tuples and accessing them in an associative way via pattern matching. By retrieving information in an associative way, Linda-like tuple spaces support *spatial decoupling* as well. Existing tuple-based coordination infrastructures like JavaSpaces ([5], [31]) or TSpaces [22] extend the capabilities of the Linda model towards the event model [19] by providing a notification mechanism. In component-based systems, notifications are generally used to observe component changes. The notification mechanism allows coordination entities to claim their interest in receiving information about specific events occurred in the coordination medium.

The space-based paradigm implemented in Corso [9] goes a different way in extending the blackboard model. It can address objects located in the space directly via its object IDs. This improves scalability and assists at garbage collection. The difference between Corso and most other space-based computing implementations is that Corso is based on peer-to-peer concepts. From the programmer’s view, the application component communicates via the space, which can virtually be accessed in a centralized way. The data itself, however, is physically distributed and replicated among the participating peers. This implies that the coordination medium is not restricted to a single server; it is rather distributed to and shared by all participating clients. The use of redundant components by maintaining several copies of data on different computers allows to continue work if relevant nodes in the system fail. Replication improves (1) *performance* by letting users access several nearby replicas avoiding unnecessary remote data calls, (2) *availability* through granting access to the data even when some of the replicas are unavailable ([8], [15]), and (3) *high fault tolerance* via redundancy [6] of the replicas.

One main problem of SBC and the reason why industry still does not widely adopt this technology seems to be lack of standards needed to make applications and SBC middleware of different vendors interoperable.

4. Solution approach and case study

In the solution approach, we present a prototype we developed for an insurance company as an answer to their requirements described in chapter 2.

4.1. Selection of coordination model

“A good distributed system should easily connect users to resources; it should hide the fact that resources are distributed across the network; it should be open; it should be scalable.” ([18], page 4) From the coordination models described in section 3, we selected Corso for the imlementation of the prototype because it sufficiently covers these points and supports most valuable features for simple realization of the requirements described in chapter 2.

Monitoring: Exploiting the characteristics of blocking read operations and of the notification mechanisms the specified shared data objects can be observed automatically in the background. The gained information can be analyzed by an independent monitoring component and its results stored in another shared data object. This object is accessible from anywhere in the network allowing managers to supervise their agents at any time.

Load balancing: Descriptions of peer groups are stored in shared data objects in the space helping to identify and to create peer groups. Those descriptions or profiles can be embedded into the notification mechanisms identifying the group of agents a stored task is meant for. Furthermore, it is possible to reassign a peer to a work item at any time by updating the work item in the Corso space. This allows starting execution of a task by a peer but finishing by another, optionally of a different peer group.

Mobile client support: The SBC technology supports members of mobile environments who permanently change their offline and online status. In fact, most of the current systems cannot handle the situation in which the requester is not available after it has sent the request. The Corso technology could transparently store the answer and make it available when the peer is available again even if the same network is accessed through new peers (e.g. with a new IP address) from different geographical positions. In Corso, distributed data structures can be designed and various replication protocols can be applied enabling selective and efficient replication.

Performance and fault tolerance: The P2P support of Corso enables application design without a central, coordinating server component. The absence of a server as single point of failure and bottleneck improves throughput, work item latency and fault tolerance.

4.2. Coordination of agents

An application that is predetermined for distribution of all kind of objects should be easy and simple to handle. It should be as transparent as possible meaning that the users of the system should only have to know who to supply the system respectively how to grab a work package. Users on the one hand are the ones who want to distribute packages and on the other hand the ones who fetch them.

This leads to the "principle of the two ears" (see Figure 2). On the left "ear" of the circle the master user provides new objects into the space meant for distribution. On the right "ear" independent users spread all over the network have to choose the desired package and start the application associated with it.

In the following the three main points of the circle are picked out and described in more detail to get a first idea how the Shared Virtual Space Distribution Manager (SVSDM) prototype works. These would be importing, distributing and collecting the packages.

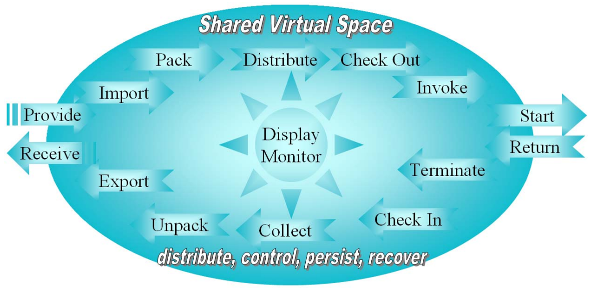

Figure 2. Functions of the SVSDM

4.2.1. Importing a package. In order to be able to distribute anything, the system needs data. This information is provided by the user of the system. This could be done either through a GUI or via files. The SVSDM prototype works with files. The user has to specify which files to distribute. This offers on the one hand independence from other systems, and on the other hand processes can work automatically with SVSDM through its offered interfaces.

The given data is zipped first and put into a communication object. Another communication object, the package, is created and tagged with additional information necessary to specify the characteristics of the package itself.

The task of the distribution manager is to put this package into the virtual space. It should be done in a way so that any other user connected to the space is informed about any updates as soon as possible.

4.2.2. Distribution of a package. Once the user decides to open a package for distribution, the system copies the package into a special list existing in the space. This list offered by Corso is the notification service. Every time the user puts a package into that list, Corso informs every other host about the new package automatically and replicates it to those sites.

At that time all available packages are visible at the worker's site. Actually, that should not be the case. The problem is that in the current version of SVSDM the usage of profiles is not fully implemented. As mentioned above, a package contains additional information. A part of this information should be used to specify the profile of a package. The user should be able to determine what kind of groups or single users are allowed to see that package in the global space. This means that some of the users do not even see a newly added package. This method does not only distribute packages within the system, it also distributes them to authorized users as well.

4.2.3. Invoke a package. At this point most of the workers should be informed about the newly added package. Depending on the network, distribution should not take more than a few seconds. The presented information of the available packages shown on an information board should help the user to select the most accurate piece of work. If the user has found the desired package of choice, she/he has to ask the SVSDM to "move" it to the user's local space. This is necessary since offline working modus should be supported as well. Once SVSDM replicated the package and all of its content to the user's site, the global space is updated. This means that on the one hand the package becomes invisible for any other user, and on the other hand the user who created the package is informed about the user who selected it.

4.2.4. Execution of the content of a package. After the work has been replicated to the user's site, she/he should have the opportunity to go offline as well. From now on, it is up to the application what happens to the content of the package.

4.2.5. Collecting a package. In some cases it is not enough to just distribute a package. A number of situations require that the output the application produces is sent back to the global space. This means that there should be the possibility to perform this task both manually and automatically via given interfaces.

Once a package has been uploaded, it cannot be selected once again. It is declared as done and removed from the global space. The answer is then stored in the local space of the initiator.

4.2.6. Exporting a package. The initiator can choose between two possibilities. She/He either deletes the package from the local space or saves the response. In the first case, any answer coming from the worker process is ignored. In case of an export, the received zipped data is saved to a user specified directory. After the data has been stored, the user still has the possibility to remove the package. Once a package has been removed from the space, it is lost forever.

4.2.7. Exception Handling. There are two kinds of ways why an exception can occur. Either the problem is related to the network or an error message appears because of a user error. Basically, Corso is able to mask network problems. If Corso for instance tries to get a primary copy of a communication object, it has to exchange messages with other peers. If the network connection breaks during the communication and has been re-established within a predefined amount of time, then the application is not notified about anything and can keep on working with the primary copy. However, if the time interval has elapsed, the SVSDM informs the application by throwing a timeout exception due to unreliable network connection. The second difficulty is concerned with the question when packages are allowed to be fetched and executed by the users. The following points represent rules a user of the SVSDM prototype has to know about:

1. A package cannot be fetched twice at the same time. If two or more users try to get a single package, all of them, except the first, will receive an error message. The first-come-first-served principle is used.
2. Over a longer period it is possible to select a package twice or even more often. In this case, the answer of the one who fetched the package last is valid. Any other responses are ignored.
3. A user is not authorized to receive packages. In that case she/he will receive a message requesting to ask for authorization first.
4. If the user responses to the content of a package that is not available any more, she/he will receive an error message.

As mentioned in the beginning of this chapter, the SVSDM is implemented and works in a way that requires only a minimum of effort on behalf of the user.

Figure 2 presents the SVSDM prototype in a way how it should be seen at a global point of view. The application gateways on the left and the right site represent the "ears" and provide access to the core SVSDM via its interfaces. Distribution of packages is done via the Distribution Manager placed between them. Behind all these Corso is situated in order to make sure that each action is performed transactional and without bothering any independent process in case of failures.

4.3. Grouping of agents

In the case description it was shown that the distribution from one client to another is not necessarily the best option. To solve this problem some additional assignment features (profile management) will be discussed. In the case of the prototype, the insurance company only wanted to test the strength of the distribution algorithm, and so complex assignment features were skipped on purpose.

4.3.1. Simple profile. The simplest way to achieve an assignment is to name the receiver. This way was implemented in the prototype. Each packet is destined for one specific agent that will be named by a personal ID in the packet.

4,3.2. Hierarchical profile. Company organizational structures are analogous to hierarchical file systems, so the representation of the workers' hierarchy may be set up in a similar manner and be built into the distribution system. This would start like a file system with a root node. Work packets may then be posted to one specific worker telling the path in the company's hierarchy. In the case of sending a package to a logical node representing a higher position within the company, the packet might be forwarded to all agents available below the given node name.

4.3.3. Complex profile (semantic web). Packets could be automatically assigned to workers based on worker profiles that express workers' skills. Profiles could be created by workers, supervisors or automatically by the system by observation of worker behavior. Each packet would carry predefined attributes that were set up at the packet's creation time. Both property files and user's skills will have to be compared and those users best fulfilling the demands should be able to download a certain packet. Solutions for similar problems are sought in many semantic web [17] projects.

5. Discussion over comparison

The main purpose of the prototype was to identify new technologies and use one of them to implement a prototype that realizes the requirements described in chapter 2. Compared to the existing solution the new prototype realizes the following improvements:

Efficient monitoring: Usage of notification mechanism supports forwarding of monitoring entries automatically. Therefore, by means of the new coordination approach, statistics for the supervisor are created automatically, and managers or supervisors can easily find out about the reliability and sedulity of their agents.

Load balancing: Work groups become more efficient, i.e. can work on more work items per time unit due to more efficient load balancing achieved through a better task distribution. This implies that no worker becomes idle, because new work tasks are in the work pool of the group and not assigned strictly to a worker.

Mobile client support: The SVSDM provides clients with replicas of work items they need for working offline. Replicas are synchronized automatically and efficiently in the background whenever network connectivity is given. In contrast, in the former solution mobile users had to connect to the server and initiate synchronization manually. The new solution thus provides mobile users with more up-to-date data with less user effort.

Performance and fault tolerance: The delay between the incoming work item and the time to start work becomes smaller as the new coordination approach removes the manager as bottleneck of the system. The delay between starting work and finishing it becomes smaller in average as agents rejecting work tasks due to heavy overload do not need to be reassigned via the server but by the group instead. The first agent that is available may take over. Fault-tolerance is supported in this sense as well since an agent may reject a work item any time. By declaring the task as high priority one of the group members may finish the task without to risk missing any deadlines specified by the task.

6. Conclusion

Motivated by the fact that future mobile information systems will face an increased demand for coordination and monitoring mechanisms in order to keep respectively to improve the service quality, we proposed the usage of a new type of technology called space-based computing. Hence, it was our objective to discuss and evaluate representative technologies with respect to a scenario coming from an insurance company, in which coordination and monitoring problems occurred. The scenario was described, and four suitable technologies were evaluated. We have shown that the space-based computing technique is a powerful tool if it is used for group communication, collaboration, and monitoring of it. The comparison of the current system of the use case scenario with the implemented prototype by means of requirements concerning *monitoring*, *load balancing*, *mobility support*, *efficiency* and *fault tolerance* (detailed in chapter 2) showed that further improvement of the prototype and the SBC technology is justified.

References

[1] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, M. Hauswirth, and S. Haridi: *The essence of P2P: A reference architecture for overlay networks*, in P2P2005, The 5th IEEE International Conference on Peer-to-Peer Computing, 2005.

[2] S. Androutsellis-Theotokis and D. Spinellis: *A survey of peer-to-peer content distribution technologies*, ACM Comput. Surv., 36(4):335–371, 2004.

[3] M. Bortenschlager: *HUG CorA - How to Use a Generic Coordination Architecture in Pervasive System Development,* In Adjunct Proceedings of the 4th

International Conference on Pervasive Computing, Dublin, 2006.

[4] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli: *MARS: a Programmable Coordination Architecture for Mobile Agents*, IEEE Internet Computing, volume 4, number 4, pages 26-35, 2000.

[5] E. Freeman, S. Hupfer, and K. Arnold. JavaSpaces: *Principles, Patterns, and Practice*, The Jini Technology Series. Addison-Wesley, 1999.

[6] F. C. Gärtner: *Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments*, ACM Computing Surveys, Vol. 31, No. 1, March 1999.

[7] D. Gelernter and N. Carriero: *Coordination languages and their significance*, Communications of the ACM, 35(2): 97–107, Feb. 1992.

[8] Hagen Höpfner and Kai-Uwe-Sattler: *Semantic Replication in Mobile Federated Information Systems*, in Proc. of the 5th Int. Workshop on Engineering Federated Information Systems (EFIS 2003).

[9] eva Kühn, How to *Approach the Virtual Shared Memory Paradigm*, In: Journal of Parallel and Distributed Computing Practices, Nova Science Books, September 1998, Vol. 1, No. 3.

[10] W. Kurschl: *Space-Based versus Message-Passing Communication A Comparison*, Technical Report TR.2004.01, Upper Austria University of Applied Sciences, 2004.

[11] A. L. Murphy, G. P. Picco, and G.-C. Roman: *LIME: A Middleware for Physical and Logical Mobility*, in F. Golshani, P. Dasgupta, and W. Zhao, editors, Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, AZ, USA), 2001.

[12] J. Newmarch: *A Programmer's Guide to Jini Technology*, Springer-Verlag, 2000.

[13] G. A. Papadopoulos and F. Arbab: *Coordination models and languages*, Advances in Computers, 46(The Engineering of Large Systems):329–400, August 1998.

[14] A. S. Rao and M. P. Georgeff: *BDI-agents: from theory to practice*, in Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco, 1995.

[15] Ratner, D., Reiher, P., Popek, G.J., Kuenning: *Replication Requirements in Mobile Environments*, G.H, Mobile Network Applications 6, 2001.

[16] K. Rehrl, M. Bortenschlager, S. Reich, H. Rieser, and R. Westenthaler: *Towards a Service-Oriented Architecture for Mobile Information Systems*, in IFIP TC8 Working Conference on Mobile Information Systems (MOBIS), pages 37–50, 2004.

[17] Stuckenschmidt, Heiner; Frank van Harmelen: *Information Sharing the Semantic Web.* Springer-Verlag, Berlin Heidelberg, 2005.

[18] Tanenbaum, Andrew S., Steen, Maarten van, *Distributed Systems Principles and Paradigms,* Prentice Hall Inc, 2002.

[19] M. Viroli, A. Ricci: *Tuple-Based Coordination Models in Event-Based Scenarios*, in Proceedings of the 22nd International Conference on Distributed Computing Systems, pages: 595-601, 2002.

[20] H. Weigand, F. van der Poll, and A. de Moor: *Coordination through Communication*, in 8th International Working Conference on the Language-Action Perspective on Communication Modeling (LAP 2003), Tilburg, The Netherlands, 2003.

[21] M. Wooldridge: *Agent-based Software Engineering*, in IEEE Proc. of Software Engineering, 144(1):26–37, Feb 1997.

[22] P.Wyckoff, S.W. McLaughry, T. J. Lehman, and D. A. Ford: *T Spaces*, IBM Journal of Research and Development, 37 (3- Java Technology):454–474, 1998.

[23] e-Business W@tch: *The European e-Business Report. A portrait of e-business in 10 sectors of the EU economy.*, 2005, retrieved on the 14th of April 2006 from http://www.ebusiness-watch.org.

[24] EITO: *European Information Technology Observatory 2006*, retrieved on the 20th of November 2006 from http://www.eito.com.

[25] *Prisma*, retrieved on the 13th of April 2005 from http://www.silverstroke.de/prisma/.

[26] S. Franklin: *Coordination without Communication*, 1996; retrieved on the 10th of January 2006 from http://www.msci.memphis.edu/franklin/coord.html.

[27] FIPA: *The Foundation for intelligent physical Agents*, 2003; retrieved on the 14th of April 2006 from http://www.fipa.org/.

[28] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa: *JADE - A White Paper*, 2003; retrieved on the 14th of April 2006 from http://jade.tilab.com.

[29] Bernhard Angerer: *Space-Based Computing*, downloaded on the 21st of November 2004 from http://www.onjava.com/pub/a/onjava/2003/03/19/java_spaces.html.

[30] *The Broker Pattern*, retrieved on the 26th of November 2006 from http://vico.org/pages/PatronsDisseny/Pattern%20Broker/.

[31] Eric Freeman: *Make room for JavaSpaces*, retrieved on the 1st of September 2004 from http://www.javaworld.com/javaworld/jw-11-1999/jw-11-jiniology.html