

Evaluating Multi-Agent System Architectures: A case study concerning dynamic resource allocation

Paul Davidsson Stefan Johansson

Department of Software Engineering and Computer Science,
Blekinge Institute of Technology, Soft Center, 372 25 Ronneby, Sweden
{pdv, sja}@bth.se,

Abstract. Much effort has been spent on suggesting and implementing new architectures of Multi-Agent Systems. However, we believe the time has come to compare and evaluate these architectures in a more systematic way. Rather than just studying a particular application, we suggest that more general problem domains corresponding to sets of applications should be studied. Similarly, we argue that it is more useful to study the properties of classes of multi-agent system architectures than particular architectures. Also, it is important to evaluate the architectures in several dimensions, both different performance-related attributes, which are domain dependent and more general quality attributes, such as, robustness, modifiability, and scalability. As a case study we investigate the general problem of "dynamic resource allocation" and present four classes of multi-agent system architectures that solve this problem. These classes are discriminated by their degree of distribution of control and degree of synchronization. Finally, we instantiate each of these architecture classes and evaluate, through simulation experiments, how they solve a concrete dynamic resource allocation problem, namely load balancing and overload control of Intelligent Networks.

1 Introduction

Much effort has been spent on suggesting and implementing new architectures of Multi-Agent Systems (MAS). Unfortunately, this work has been carried out in a quite unstructured way where a (group of) researcher(s) invents a new architecture and applies it to a particular domain and conclude that it seems to be appropriate for this domain. Often this new architecture is not even compared to any existing architecture. We believe that this area now has reached the level of maturity when it is appropriate to compare and evaluate these architectures in a more systematic way.

1.1 Applications

Of course, there is no single MAS architecture that is the most suitable for all applications. On the other hand, to find out whether one architecture performs better than another for a particular application is usually of limited scientific interest. (Although this information may be very useful to solve that particular problem.) Instead, we suggest the study of more general problem domains corresponding to sets of applications

with common characteristics. In this paper we will exemplify this approach by investigating the general problem of *dynamic resource allocation*. Such studies tend to be quite abstract and are for that reason often of a theoretical and qualitative nature. They therefore should be supplemented with and validated by quantitative empirical studies in one or more concrete applications corresponding to instances of the general domain, in this paper exemplified by load balancing and overload control in *Intelligent Networks*, a type of telecommunication system.

1.2 Architectures

Just as it is useful to study classes of applications rather than particular applications, we argue that it is useful to study classes of MAS architectures in addition to particular architectures. To make such studies possible, we need to describe MAS architectures in a way that abstracts the particularities of the individual architectures but still captures their relevant characteristics. To develop a general way of characterizing MAS architectures is in itself a major research task. In fact, it may be necessary to use several views to capture all relevant aspects of an architecture cf. Kruchten [9].

In this work we will categorize MAS architectures according to two properties: the type of control used (from fully centralized to fully distributed), the type of coordination (synchronous vs. asynchronous). As with classes of applications, it is mostly theoretical studies that can be performed on classes of MAS architectures. Therefore, they often need to be supplemented with empirical studies using instantiations of these architectures. Below we will present four concrete architectures corresponding to different combinations of the two architectural properties.

1.3 Quality attributes

It is possible to evaluate MAS architectures with respect to several different quality attributes, both different performance-related attributes and more general quality attributes, such as, robustness, modifiability, and scalability. Some of these attributes are domain independent and some are specific for each set of applications, e.g., performance-related attributes. As mentioned earlier, we do not think it is possible to find a MAS architecture that is optimal with respect to all relevant attributes. Rather, there is an inherent trade-off between these attributes and different architectures balance this trade-off in various ways. The various applications, on the other hand, require different balances of this trade-off. Thus, in order to choose the right architecture for a particular application, knowledge about relevant attributes and how different MAS architectures support them is essential.

1.4 Evaluation framework

To evaluate a set of architectures in a systematic way, we suggest an approach that can be described in terms of the following three-dimensional space:

- the set of possible applications,
- the set of possible MAS architectures, and

- the set of attributes used to evaluate the architectures.

The suggested approach is to investigate substantial parts of this space rather than just single points. We believe that this approach, besides of enabling a more systematic investigation of the space, will lead to a deeper understanding of MASs and their applications, which, in turn, will contribute to reach the long-term goal of obtaining general design principles of MASs.

In this paper we will apply this approach to the general problem of dynamic resource allocation and present four abstract MAS architectures with different characteristics that solves the problem. These will then be compared with respect to a number attributes, e.g., reactivity, ability to balance the loads, fairness, utilization of resources, responsiveness, amount of communication overhead, robustness, modifiability, and scalability. Finally, we evaluate concrete instantiations of these abstract architectures in a concrete dynamic resource allocation problem, namely load balancing and overload control in Intelligent Networks.

2 Abstract domain: Dynamic resource allocation

Multi-agent technology has proved to be successful for *dynamic resource allocation*, e.g. power load management [11] and cellular phone bandwidth allocation [3]. Basically, this problem concerns the allocating of resources between a number of *customers*, given a number of *providers*. The dynamics of the problem lies in that the needs of the customers, as well as the amount of resources made available by the providers, vary over time. The needs and available resources not only vary on an individual level, but also the total needs and available resources within the system. We will here assume that the resources cannot be buffered, i.e., they have to be consumed immediately, and that the cost of communication (and transportation of resources) between any customer-provider pair is equal.

2.1 Abstract multi-agent architectures

There are many ways of dividing the set of possible MAS architectures into different subsets based on their characteristics, e.g.:

- the topography of the system,
- the degree of mobility and dynamics of the communications,
- the degree of distribution of control, and
- the degree of synchronization of interaction.

We have chosen to focus the two last properties. By degree of *distribution* we mean to what degree the *control* of the system is distributed. The degree of *synchronization* is a measure of how the execution of the agents interrelate with each other. We may have agents that are highly sophisticated, but who only interacts at special slots in time, and thus have a high degree of synchronization. There are also systems in which the agents may interact continuously, independently of when other agents interact, which we will refer to as *asynchronous*.

To sum up, we will compare the following four abstract classes of MAS architectures for dynamic resource allocation: centralized synchronous architectures, centralized asynchronous architectures, distributed synchronous architectures, and distributed asynchronous architectures.

2.2 Abstract attributes

We have identified the following important performance-related attributes to dynamic resource allocation:

- Reactivity: *How fast are resources re-allocated when there are changes in demand?*
- Load balancing: *How evenly is the load balanced between the resource providers?*
- Fairness: *Are the customers treated equally?*
- Utilization of resources: *Are the available resources utilized as much as is possible?*
- Responsiveness: *How long does it take for the customers to get response to a request?*
- Communication overhead: *How much extra communication is needed for the resource allocation?*

In addition, there are a number of more general software architecture quality attributes [6] that should be addressed, e.g.:

- Robustness: *How vulnerable is the system to node or link failures?*
- Modifiability: *How easy is it to change the system after it is implemented (and often deployed)?*
- Scalability: *How good is the system at handling large numbers of users (providers and customers)?*

2.3 Theoretical/qualitative evaluation

We will now make a brief theoretical, or qualitative, analysis of how the degree distribution and synchronization of the multi-agent system architecture influence the attributes identified in the last section.

- Reactivity should be promoted by asynchronous architectures since there is no need to await any synchronization event before i) an agent can notify other agents about changes in demand and ii) other agents can take the appropriate actions to adapt to these changes.
- Load balancing should be favored, or at least not disfavored, by centralized control since it is possible to take advantage of the global view of the state of the system, e.g., the current load at the providers and the current demand of the customers.
- Similarly should fairness be easier to achieve for architectures with centralized control since they have information about the global state of the system.
- It is not clear from a strictly theoretical analysis if there is any correlation between the ability to utilize the resources and the architectural properties. Empirical studies are probably necessary.

- Also, it is not clear from a strictly theoretical analysis if there is any correlation between responsiveness and the architecture properties.
- Communication overhead can be measured either by the number of messages sent, or by the bandwidth required for the allocation. Synchronous architectures tend to concentrate the message sending to short time intervals, and thus requiring a large bandwidth, whereas asynchronous architectures tend to be better at utilizing a given bandwidth over the time. Also, communication in distributed architectures has a tendency to be more local than in centralized architecture, using smaller parts of the network.
- Regarding robustness we conclude that the more centralized the control is, the more vulnerable the system gets. Basically, the reason for this is that the system cannot function, i.e., perform reallocation, if the agents that are responsible for the control fail. In more distributed systems, the reallocation may function partially even though some agents have failed.
- The modifiability, to add or remove a provider or customer, seems to be better in centralized architectures. For instance, changes may only be necessary in one part of the system.
- Scalability seems to be better supported by distributed architectures than centralized architectures. Firstly, the computational load for the resource allocation is divided between a number of computers, and secondly, the risk for communication bottlenecks is smaller.

3 Concrete domain: Load balancing in Intelligent Networks

One important area in which the dynamic resource allocation problem is present is telecommunications. The Intelligent Network (IN) concept was developed in order to enable operators of telecommunication networks to create and maintain new types of services [10]. Two important entities of an IN are the Service Switching Points (SSPs) and the Service Control Points (SCPs). The SSPs continuously receive requests of services which they cannot serve without the help of the SCPs where all service software resides. Thus, the SCPs are providers and the SSPs are customers. The SSPs and SCPs communicate via a signaling network which we here will represent as a *cloud* rather than a specific topology of signaling links and nodes. (See Figure 1.) It is assumed that a small part of the available bandwidth of this network is reserved for the resource allocation, i.e., the communication overhead caused by agent communication (and transportation). It is assumed that all SCPs support the same set of service classes and that all service requests can be directed by a SSP to any SCP.

3.1 Concrete multi-agent system architectures

We have chosen one architecture of each of the abstract classes mentioned earlier (see table 1). Common for these architectures are the use of three different types of agents: *quantifiers*, *allocators*, and *distributors*. A quantifier acts on behalf of a provider of the resources, an allocator acts on behalf of a customer, and a distributor decides the allocation of some (or even all) available resources. Although these three types of agents

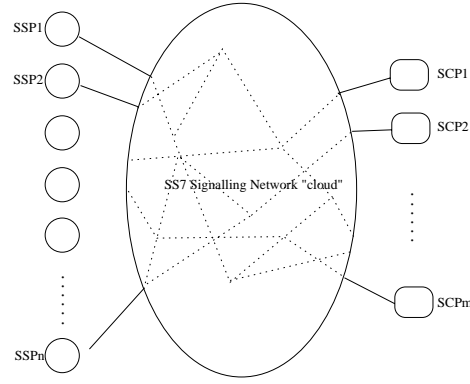


Fig. 1. A simplified view of an Intelligent Network (IN) with m SCPs and n SSPs (typically $n > m$).

	centralized	distributed
synchronous	Centralized auctions (CA)	Hierarchical auctions (HA)
asynchronous	Centralized leaky bucket (CLB)	Mobile brokers (MB)

Table 1. The four different multi-agent system architectures classified in terms of distributedness and synchronicity

have similar roles in all the four multi-agent system architectures, the actual implementation may be rather different (in particular this hold for the distributors). The reason, of course, is that different system architectures may put different demands on the agents.

The centralized auction architecture The *Centralized Auction* (CA) architecture is an example of a synchronous, centralized architecture. Arvidsson et al. [1] suggested an approach where the resource allocation is carried out by means of tokens (cf. market-based control [5]). Each token represents a service request and is consumed when the request is accepted by a provider. The three types of agents have the following functionality:

- The quantifiers try to *sell* the amount of tokens that corresponds to the load that the provider is able *serve* between two auctions.
- The allocators try to *buy* the amount of token corresponding to the resources it predicts their customer will receive during the time to the next auction.
- The distributor receives bids from the quantifiers corresponding the available capacity at their provider (and the *prices*), and bids from the allocators containing the expected need for resources. The distributor then carries out the auction so that the common good is maximized and sends messages about the result to the involved agents.

An allocator maintains a pool of tokens for each provider and type of resource. Each time the allocator feeds a provider with a request for a particular type of resource, one token is removed from the associated pool. If all pools associated with a particular resource type are empty, the customer cannot accept more requests. The pools are refilled at the auctions that take place at fixed time intervals. In order to avoid spending all tokens immediately during high loads (which would lead to excessive delays caused by long queues at the providers), percentage thinning is used so that the probability of buying a certain type of resource is never higher than the number of remaining tokens over the number of expected needs during the remainder of the interval. For more details we refer to Arvidsson et al. [1].

The hierarchical auction-based architecture One possible implementation of a distributed, synchronous system is the *hierarchical auction* (HA) architecture [11]. The idea is to partition the set of allocators and to use one distributor for aggregating bids and holding auctions for each partition. These distributors then connect to higher order distributors in a hierarchical manner until the total demand can be matched against the amount of available resources offered by the quantifiers.

The centralized leaky bucket architecture The centralized asynchronous architecture we have chosen is based upon an asynchronous approach called *Leaky bucket* [2]. The basic idea is that each provider is equipped with a Leaky bucket that feeds requests to the provider at an even and optimal rate. This is done by inserting the incoming requests from the customers in a queue in the Leaky bucket. These requests are then dequeued at a rate corresponding to the maximum capacity of that provider. If the queue is full, the requests are rejected.

To get a *centralized* architecture, we introduce a *centralized leaky bucket* (CLB) architecture, in which there is just one central distributor, common for all allocators and quantifiers. The allocators send all requests immediately to this distributor, which consists of a common leaky bucket for queuing the requests. It also has a router that continuously dequeues requests at a rate corresponding to the total capacity of the providers and then forwards the requests evenly to the providers in proportion to their capacity. If the bucket is full, the request is returned to the allocator where it is rejected.

The mobile broker architecture As an example of a distributed, asynchronous system, we choose a *mobile broker* (MB) architecture [4]. In this architecture, the distributors are implemented as mobile brokers (one for each provider) that sequentially visit each (or a subset) of the allocators offering the resources currently available at the corresponding provider. The allocator then *requests* the resources it needs for the moment (or rather, predicts it will need in the near future). If possible, the broker gives this amount of resources to the allocator. Otherwise, it gives as much as is currently available at the provider. However, there are two problems with this naive approach:

- If an allocator demands all the available resources, the broker will give them to that allocator. Thus, the broker will not be able to hand out any more resources for a while, which would not be fair.

- If the overall load is low or moderate, the allocators are given just as much resources as they demand. However, if an allocator need slightly more resources than it asked for (predicted), it will have to turn down request, even though the provider has lots of surplus capacity.

In order to solve these problems, we use a broker mechanism that strive to give out all the available resources and give each allocator resources in proportion to their part of the total current demand (of the allocators in the route). For the details of this approach we refer to Johansson et al. [7]. It should be noted that in case of a sudden increase in demand, the resources given out may momentarily exceed the available resources, which in the worst case will lead to a transient overload situation. However, the mechanism is self-stabilizing, and will find an equilibrium within one route (given that the demands are relatively stable). The mechanism ensures that the allocators are given resources that (relatively) correspond to their share of the total demand handled by the broker, thus solving both problems in the naive solution above.

If an allocator are visited by several brokers it may happen that some of the brokers' SCP are carrying a higher load than the others. To deal with this problem an additional balancing function is used, making the allocators try to move load from those SCPs with relatively high load to those with relatively low load. The allocator calculates the load of a broker from the quotient between what it asked for and what is was given by the broker.

3.2 Concrete attributes

We now operationalize the abstract attributes presented earlier. Thus, in the domain of IN load management the attributes are defined as follows:

- Reactivity is measured by how fast the MAS is able to re-allocate the available SCP processing time when there are sudden changes of offered loads by the SSPs.
- Load balancing is measured by the standard deviation between the carried load of the SCPs.
- Fairness is measured by the standard deviation of rejected calls divided by the accepted calls between the SSPs, i.e., the rejection rate.
- The utilization of resources is measured by how close the carried load is to the target load, or offered load, if the offered load is less than the target load. SCP load levels should be as close to the target load (e.g., 0.9 Erlangs, corresponding to 90% of its capacity) as possible but not exceed it. If an overload situation is approaching, the SSPs should throttle new requests.
- Responsiveness is measured by the time it takes for the SSPs to get response from an SCP.
- Communication overhead is measured by the bandwidth necessary for the MAS to perform the reallocation.
- Robustness is measured in terms of the consequences of a distributor agent failure.
- Modifiability is measured in terms of how easy it is to add a new or remove an existing SSP or SCP.
- Scalability is measured in terms of how the number of SSPs and SCPs influence the performance-related attributes.

3.3 Experimental evaluation

The four concrete architectures have been evaluated in simulation experiments. For these experiments we used the same simulation model as Arvidsson et al. [1]. The Intelligent Network modeled has 8 SCPs and 32 SSPs, which are assumed to be statistically identical respectively. The processors at the SCPs are engineered to operate at 90 percent of the maximum capacity (target load is 0.9 Erlang). All messages passing through the network experience a constant delay of 5 ms, and it is assumed that no messages are lost. Detailed descriptions the simulation results can be found in [8]. Since we use these experiments as a case study, we only summarize the results here:

- As the theoretical evaluation predicted, reactivity was promoted by the two asynchronous architectures. For instance, we simulated a scenario in which half of the 32 SSPs experience an offered load corresponding to 0.2 Erlang, and the other half a load corresponding to 1.4 Erlang. The whole system thus is offered a total average load of 0.8 Erlang, which is below the target load, and therefore possible for the system to carry. At time 400, the levels of loads are shifted from high to low and vice versa and 10 seconds later they are swapped back again, see Fig. 2. The CLB manage this difficult situation perfectly and the other asynchronous architecture MB does almost as well. The auction-based architectures, on the other hand, have apparent difficulties to adapt to the changes.
- All the architectures were able to balance the load well. There were just small differences in the standard deviations (of the measured carried load of the SCPs), varying between 1 and 3 mErlang depending on the offered load. When the offered load was below target load the centralized architectures performed slightly better, whereas there were no measurable differences in overload situations.
- With respect to fairness, all architectures performed well when the offered load was below target load. In overload situations, MB was significantly less fair than the other architectures. However, it may be the case that an improvement of the design of the broker routes may reduce these differences.
- It was not clear from theoretical analysis if there is any correlation between the ability to utilize the resources and the architectural properties. The simulation results presented to the left in Fig. 3 indicates that centralized asynchronous architectures perform best in this respect.
- Also, it was not clear the theoretical analysis if there is any correlation between responsiveness and the architecture properties. But in this case the simulation results indicate that the centralized asynchronous architecture has the worst performance. See Fig. 4.
- We tuned the parameters in the simulation experiments so that both CA, HA and MB needed approximately the same bandwidth for communication overhead, about 40 messages/second irrespective of the amount of offered load. The bandwidth needed by CLB, however, is proportional to the number of requests and is considerably higher than for the other architectures. For instance, when the offered load is 0.70 Erlang, 2150 messages/second are sent, and when the offered load is 2.0 Erlang, 9365, i.e., more than 200 times as much bandwidth as the other architectures need.

For the general software architecture attributes (robustness, modifiability, and scalability), we refer to the theoretical analysis in section 2.3.

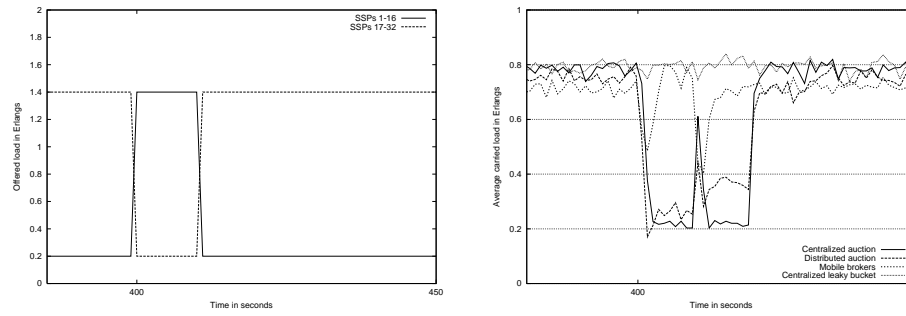


Fig. 2. The expected offered load (left) and the average carried load (right).

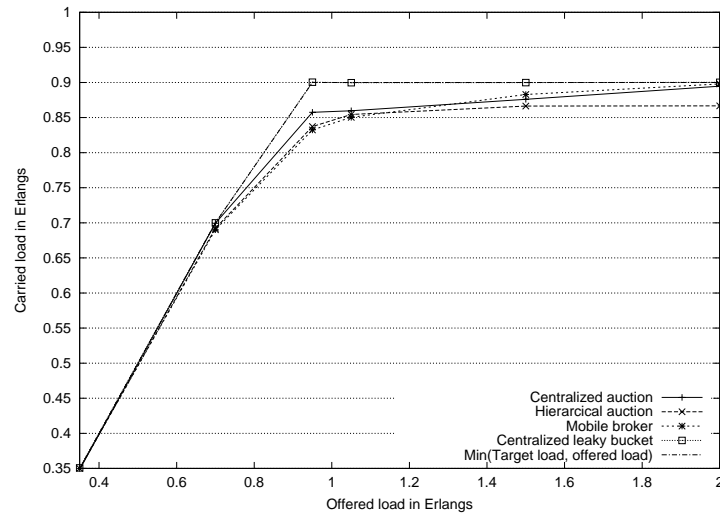


Fig. 3. The ability to carry offered when using the different architectures.

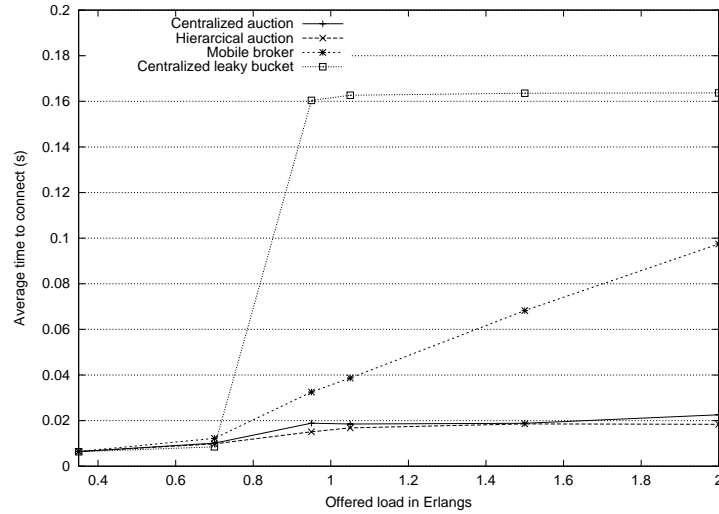


Fig. 4. The average response time for the connections when using the different architectures

4 Conclusions and future work

We have described a systematic way of evaluating different aspects of different MAS architectures. This approach was applied to an abstract domain, namely dynamic resource allocation, and a theoretical evaluation of abstract architectures was made. This was supplemented by an experimental study of an implementation of this abstract domain, load balancing and overload control in Intelligent Networks. The experimental evaluation confirmed the conclusions of the theoretical analysis, e.g., that asynchronous architectures are able to react faster than synchronous. In addition, it gave insights concerning attributes for which no clear conclusions could be achieved from the theoretical analysis, e.g., that centralized asynchronous architectures are able to utilize the available resources better than the other architectures, but have larger delays and need more bandwidth when the load is high.

The results of the case study were, not very surprisingly, that different architectures excel in different dimensions. The choice of MAS architecture for a particular application should be guided by the balance of the trade-off between these dimensions that is optimal for that application. We believe that if the systematic approach suggested here is widely adopted, such choices can be more informed than is currently practice.

Our plans for future work includes:

- Further experimental validation in the IN domain of the theoretical results regarding, e.g., scalability.
- Experimental validation in another dynamic resource allocation domain.
- Use the outlined approach to investigate other domains than dynamic resource allocation.

- Investigating to what extent the implementations of the individual agents influence system performance.
- Develop a more refined method for characterizing MAS architectures.

Acknowledgments

The authors would like to thank Martin Kristell for assistance during the simulation experiments and the Swedish Knowledge Foundation for funding parts of this work.

References

1. Å. Arvidsson, B. Jennings, L. Angelin, and M. Svensson. On the use of agent technology for IN load control. In *Proceedings of the 16th International Teletraffic Congress*. Elsevier Science, 1999.
2. A.W. Berger. Comparison of call gapping and percent blocking for overload control in distributed switching systems and telecommunication networks. *IEEE Trans. Commun.*, 39:574–580, 1991.
3. E. Bodanese and L. Cuthbert. An intelligent channel allocation scheme for mobile networks: An application of agent technology. In *Proceedings of the 2nd International Conference on Intelligent Agent Technology*, pages 322–333. World Scientific Press, 2001.
4. B. Carlsson, P. Davidsson, S.J. Johansson, and M. Ohlin. Using mobile agents for IN load control. In *Proceedings of Intelligent Networks '2000*. IEEE, 2000.
5. S.H. Clearwater, editor. *Market-Based Control: Some early lessons*. World Scientific, 1996.
6. P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. Addison Wesley, 2002.
7. S.J. Johansson, P. Davidsson, and B. Carlsson. Coordination models for dynamic resource allocation. In A. Porto and G.-C. Roman, editors, *Coordination Languages and Models*, volume 1906 of *Lecture notes in computer science*, pages 182–197. Springer Verlag, 2000. Proceedings of the 4th International Conference on Coordination.
8. S.J. Johansson, P. Davidsson, and M. Kristell. Four architectures for dynamic resource allocation. In *Proceedings of Fourth International Workshop on Mobile Agents for Telecommunication Applications*, 2002. Springer Verlag, LNCS.
9. P.B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, pages 42–50, July 1995.
10. T. Magedanz and R. Popescu-Zeletin. *Intelligent Networks*. International Thomson Computer Press, 1996.
11. F. Ygge. *Market-Oriented Programming and its Application to Power Load Management*. PhD thesis, Lund University, Sweden, 1998.