Using the Analytic Hierarchy Process for Evaluating Multi-Agent System Architecture Candidates

Paul Davidsson, Stefan Johansson, and Mikael Svahnberg

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

Abstract. Although much effort has been spent on suggesting and implementing new architectures of Multi-Agent Systems (MAS), the evaluation and comparison of these has often been done in a rather ad-hoc fashion. We believe that the time has come to start doing this in a more systematic way using established methods. For instance, we argue that it is important to evaluate the architecture candidates for a particular application according to several quality attributes relevant to that application. The architecture that provides the most appropriate balance between these attributes should then be selected. As a case study we investigate the problem of load balancing and overload control of Intelligent Networks and present four MAS architectures that can be used to handle this task. We instantiate each of these and define metrics for the selected quality attributes. The instantiations are studied in simulation experiments and measurements of the metrics are recorded. The measurements are then analyzed using the Analytic Hierarchy Process, which is a basic approach to select the most suitable alternative from a number of alternatives evaluated with respect to several criteria. We illustrate how such analyzes can be used for deciding which architecture candidate is the most appropriate in different situations.

1 Introduction

Much effort has been spent on suggesting and implementing new architectures of Multi-Agent Systems (MAS). However, less work has been done in studying how these architectures should be evaluated and compared. Most evaluations and comparisons have been carried out in a quite unstructured way. For instance, when a (group of) researcher(s) invents a new architecture and applies it to a particular domain and concludes that it seems to be appropriate for this domain. Often this new architecture is not even compared to any existing MAS architecture. Also, the selection between candidate architectures for a particular application is typically done in a rather *ad hoc* fashion. We believe that this area has now reached the level of maturity when it is appropriate to compare and evaluate MAS architectures in a more systematic manner. In this paper, we show how an established method from Management Science can be used to achieve this, e.g., by taking into account several quality attributes and weighting them according to the requirements of the application at hand.

Of course, there is no single MAS architecture that is the most suitable for all applications. Since agent technology has shown to be successful for *dynamic resource*

allocation, e.g. power load management [22] and cellular phone bandwidth allocation [5], we have chosen this domain for our architecture style comparison. Basically, the problem concerns allocation of resources between a number of *customers*, given a number of *providers*. The dynamics of the problem lie 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 may vary over time. 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. As a concrete case of dynamic resource allocation we have chosen load balancing and overload control in *Intelligent Networks*, a type of telecommunication system.

1.1 Architectural styles

We argue that it is useful to study classes of MAS architectures, corresponding to *architectural styles* [19] in addition to particular architectures. These may describe abstractions of software entities of varying abstraction levels such as enterprise architectures, system architectures, subsystem architectures, or the architecture within a particular component, and may involve several views of the architecture to capture all relevant aspects cf. Kruchten [14].

In this work we focus on a particular abstraction level and characterize MAS architectural styles according to two properties: the type of control used (from fully centralized to fully distributed), and the type of coordination (synchronous vs. asynchronous). We compare the following four MAS architectural styles for dynamic resource allocation: centralized synchronous architectures, centralized asynchronous architectures, distributed synchronous architectures, and distributed asynchronous architectures. As the studies that can be performed on MAS architectural styles are mostly of a theoretical nature, they often need to be supplemented with empirical studies using instantiations of these styles. Below we present four concrete architectures corresponding to each of the architectural styles. This may be seen as an early attempt to construct a domain-specific system of patterns as is discussed in Chapter 5 of Buschmann et al. [7].

It should be noted that the terminology varies between different sources. For example, what Shaw & Garlan [19] refer to as architectural styles is, using Bosch's [6] terminology, a mixture of architectural patterns and architectural styles. Bosch's architectural patterns, in turn, conforms closer to Buschmann et al's [7] terminology, which also has some overlap with Shaw & Garlan's architectural styles. In this article we have chosen to use the term architectural style to refer to the abstract architectural structures that may be used as a starting point when creating a concrete architecture for a particular MAS system.

1.2 Quality attributes

It is possible to evaluate MAS architectural styles with respect to several different quality attributes [9]. Some of these attributes are domain independent and some are specific for each set of applications, e.g., performance-related 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 an individual request?*), and Communication overhead (*How much extra communication is needed for the resource allocation?*).

In addition, there are a number of more general software architecture quality factors [16] that could be addressed, e.g. Robustness and Modifiability. Although these softer quality factors are important when building real systems, we choose not to include them in this evaluation, since they are difficult to measure in quantitative terms. We will however later discuss the possibility to include factors of this kind later on in the discussion.

1.3 AHP-based evaluation

Typically, an architecture constitutes a balance between different quality attributes, just as different applications may require a specific balance or trade-off between quality attributes. Hence, to select the most suitable architecture for a particular application knowledge about relevant attributes and how different MAS architectures support them is essential.

The Analytic Hierarchy Process (AHP) [17, 18] is a multi-criteria decision support method from Management Science [1] that has previously been tried in various and similar software engineering settings (e.g. [12, 13, 20, 21]). One of the cornerstones in AHP is to evaluate a set of alternatives based on a particular blend of criteria, i.e. considering a specific trade-off situation. The AHP can quantify subjective assessments through a process of pair-wise comparisons or use tangible data e.g. from a simulation.

The first step in AHP is to set up a hierarchy of the criteria that are being evaluated. This means that one criterion can be broken down into several sub-criteria, and the evaluation of the different alternatives is done by weighing in all levels of this decision support hierarchy. In our case (outlined in Fig. 1), the top-level goal is *Most Appropriate Architectural Style*. Under this top-node in the hierarchy the different evaluation criteria are listed, in our case *Reactivity*, *Load Balancing*, *Fairness*, *Utilization of Resources*, *Responsiveness*, and *Communication Overhead*. For three of these criteria a further specialization is necessary, namely the expected load of the target system. Hence, under the criteria Utilization of Resources, Responsiveness, and Communication Overhead, we add as criteria different levels of offered loads, i.e. *0.350*, *0.700*, *0.950*, *1.05*, *1.50*, and *2.00* Erlang¹. For a particular system, the criteria are then prioritized in accordance with how desired they are for the system. This prioritization is done both for the different loads and the aforementioned quality attributes, and can be done using e.g. the pair-wise comparison process provided in the AHP method or by means of any other prioritization method. For future use, we also at this stage make sure that the priorities on a particular level in the decision tree are normalized so that they sum up to one.

¹ 1.0 Erlang correspond to 100% load.

Fig. 1. AHP Decision Support Hierarchy

This decision support hierarchy is used in conjunction with the different alternatives (the candidate architectures in the particular domain) as follows. For each of the leaf nodes in the decision support hierarchy we compare each of the candidate architectures (described in Section 2) with the other candidate architectures. This can be done by using a pair-wise comparison process or by providing tangible data. In this study, we use tangible data from a simulation (further described below).

To ensure that different measurements are comparable we first calculate their so called Z-score, Z_v where the values are normalized to be distributed around zero [10]. We then "move" these values so that the smallest value is zero, and then divide all values with the sum of the values. Ultimately, the values for a particular measurement are thus normalized so that their total sum $\sum_{i \in V} Norm_i = 1$. This process enables us to take data using any unit of measurement for each criterion and still compare the candidate architectures over a number of different criteria. The equations are thus:

$$Z_v = \frac{v - \mu_x}{\sigma_x} \quad \text{where } \mu_x \text{ and } \sigma_x \text{ are the average value and the standard deviation respectively}$$

$$M_v = Z_v + |\min_{i \in V} Z_i|$$

$$Norm_v = \frac{M_v}{\sum_{i \in V} M_i}$$

The obtained normalized values for the candidate architectures are then multiplied with the normalized priorities for each level in the decision support hierarchy (i.e. the quality attributes and the desired offered load). Lastly, the results of these multiplications are summed for each candidate architectural style. These sums represent the suitability of each alternative *in relation to the other alternatives*. It is thus not absolute numbers but a ratio compared to the other alternatives that is obtained.

2 Case Study: 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

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

Table 1. Studied MAS Architectures.

enable operators of telecommunication networks to create and maintain new types of services [15]. 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. 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.

2.1 Concrete multi-agent system architectures

We have chosen one architecture of each of the abstract classes mentioned in Table 1.

Common for these architectures are the use of three different types of agents: *quantifiers*, *allocators*, and *distributors*[2]. 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 have similar roles in all the four multi-agent system architectures, the actual implementation may be rather different (in particular this holds 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. [2] suggested an approach where the resource allocation is carried out by means of tokens (cf. market-based control [8]). 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 to *serve* between two auctions.
- The allocators try to *buy* the amount of tokens corresponding to the resources they predict their customers will receive during the time to the next auction.
- The distributor receives bids from the quantifiers corresponding the available capacity at their providers (and the *prices*), and bids from the allocators with the expected needs 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. [2].

The hierarchical auction-based architecture One possible implementation of a distributed, synchronous system is the *hierarchical auction* (HA) architecture [22]. 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 chosen is based upon an asynchronous approach called *Leaky bucket* [4]. 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, the *centralized leaky bucket* (CLB) [11] is introduced, in which there is just one central distributor, common for all allocators and quantifiers. The allocators send all requests to this distributor, 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, a *mobile broker* (MB) architecture [11] is selected. 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 requests, even though the provider has lots of surplus capacity.

In order to solve these problems, a broker mechanism is used 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. [11].

If an allocator is visited by several brokers it may be that some of the brokers' SCPs 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 higher load to those with lower 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.

2.2 Metrics

We now operationalize the quality attributes presented in Section 1.2. In the domain of IN load management the metrics 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. We measure the number of time steps it takes in the simulation between the rise in requested load from 0.35 Erlang to 0.70 Erlang and the time step when the offered load meets the requested load again.
- Load balancing is measured by the standard deviation between the carried load of the SCPs. We measure the average of the standard deviations for 500 time units.
- Fairness is measured by the standard deviation of accepted calls divided by the generated calls between the SSPs, i.e., the acceptance rates. The acceptance rates are measured for all SSPs and the standard deviation of these rates s_{ssp} is calculated. $1 - s_{ssp}$ is then finally our fairness measure.
- 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 Erlang, corresponding to 90% of its capacity) as possible but not exceed it. If an overload situation is approaching, the SSPs should throttle new requests. This is measured by taking the actual average carried load of the system.
- Responsiveness is measured by the time it takes for the SSPs to get response from an SCP.
- Communication overhead is measured by the bandwidth (in terms of number of messages per time unit) necessary for the MAS to perform the reallocation.

2.3 Experimental setup and results

The four concrete architectures have been evaluated in simulations consisting of 8 SCs and 32 SSPs. Here we use refined data from a large series of simulation experiments that provides estimations of the performance measures of the architecture candidates. Complete descriptions of the simulation results can be found in Johansson et al. [11] where all the technical details of the experiments are thoroughly described. Considering the limited space available, and as the aim here is to explain the use of AHP, we leave out most of the technical details.

		Raw values				$Norm_v$			
		CA	HA	CLB	MB	CA	HA	CLB	MB
Reactivity		12.12	12.12	2.02	4.04	0	0	0.5556	0.4444
Load Bal.		0.04937	0.04888	0.03539	0.2111	0.3237	0.3247	0.3517	0
Fairness		0.98243	0.98052	1	0.99260	0.1280	0	0.4958	0.3762
Utilization of resources	0.35	0.3484	0.3492	0.3507	0.3500	0	0.1709	0.4904	0.3387
	0.70	0.6993	0.6922	0.7000	0.6902	0.4368	0.0957	0.4675	0
	0.95	0.8575	0.8372	0.9005	0.8324	0.2562	0.0488	0.6950	0
	1.05	0.8598	0.8544	0.8997	0.8503	0.1500	0.0650	0.7850	0
	1.50	0.8760	0.8664	0.8998	0.8829	0.1608	0	0.5617	0.2775
	2.0	0.8948	0.8666	0.9003	0.8980	0.3021	0	0.3615	0.3364
Responsiveness	0.35	0.006403	0.006330	0.006409	0.006482	0.2614	0.2409	0.4977	0
	0.70	0.01010	0.00979	0.00847	0.01150	0.2283	0.2784	0.4933	0
	0.95	0.01886	0.01510	0.1604	0.02775	0.3374	0.3463	0	0.3162
	1.05	0.01849	0.01680	0.1627	0.03251	0.3431	0.3471	0	0.3097
	1.50	0.01882	0.01857	0.1636	0.06193	0.3698	0.3705	0	0.2597
	2.0	0.02257	0.01839	0.1637	0.07656	0.3778	0.3890	0	0.2332
Communication overhead	0.35	72	80	1010	85	0.3358	0.3330	0	0.3312
	0.70	72	80	2020	85	0.3345	0.3332	0	0.3323
	0.95	72	80	2740	85	0.3342	0.3332	0	0.3326
	1.05	72	80	3029	85	0.3341	0.3332	0	0.3327
	1.50	72	80	4328	85	0.3339	0.3333	0	0.3329
	2.0	72	80	5771	85	0.3337	0.3333	0	0.3330

Table 2. The raw data v and the z-score normalization, $Norm_v$, of each of the properties of the four architectures.

Property	Reactivity	Load Balancing	Fairness	Utilization	Responsiveness	Communication
Priority P_c	0.10	0.10	0.10	0.10	0.10	0.50
Priority P_u	0.20	0.20	0.10	0.30	0.20	0

Table 3. Priorities of the various properties in the case of a restricted communication (P_c) and limited resources (P_u).

The values according to the metrics defined in the last section are presented in Table 2. We include two different priorities (Table 3) in order to show how changes in priorities may change the results. It should be noted that these are *examples* of priorities and as such they are of course of limited interest in a general meaning. The actual priorities should be set for the specific system considered.

2.4 AHP analysis

Using the data from the experiment described above, we are now able to instrument the AHP decision support hierarchy with the evaluations of the architectural styles for each of the criteria and each level of desired load. The values used for this instrumentation are listed in Table 2 under the heading "Raw Values". This data is then normalized as described in Section 1.3. During this normalization we also take into account that for many metrics a low value is more desirable than a high value. We do this by multiplying the normalized Z_v score with -1 before shifting and re-normalizing the values. The raw values are inverted for all quality attributes except Fairness and Utilization of resources.

As an illustration, we provide two examples of priorities for the different quality attributes shown in Table 3. These two cases corresponds to one scenario where the (potential) system bottleneck lies in the communication network (P_c) and one where the resources (the SCPs) are the limiting factor (P_u). In the first case it is important to keep communication overhead at a low level, whereas in the second case it is not. Instead, utilization of the resources is prioritized. For three of the attributes (Utilization of resources, Responsiveness, and Communication overhead) we also need to weigh in the desired offered load, using the example values presented in Table 4. We use the same desired offered load in both example cases.

For each of the two cases we use the process described in Section 1.3 by taking the product of the priorities of the quality attributes and desired load, and multiply this with the corresponding normalized value, $Norm_v$, for each candidate architectural style. The result of this is then summed for each candidate architectural style, and presented in Table 5. As can be seen, in the first case P_c , with restricted communication abilities,

Load l	0.35	0.70	0.95	1.05	1.50	2.0
Weight w_l	0.10	0.15	0.25	0.25	0.15	0.10

Table 4. Weights of the six different levels of loads.

P_c		P_u	
MB	0.281	CLB	0.439
CA	0.267	CA	0.209
HA	0.238	MB	0.203
CLB	0.214	HA	0.150

Table 5. Results of the AHP given the two priorities P_c and P_u .

the MB architecture is the most suitable, followed by CA, HA and CLB. In the second case P_u , with restricted computing resources, the CLB architecture is more than twice as good as its nearest competitors CA and MB, and almost three times as good as HA.

3 Discussion

Architectural styles have received considerable attention in the software engineering community during the past 10 years (cf. ref. [3, 7, 6, 19]) because of the way that they capture previous experiences and extract the essentials of different architectural design solutions. One important issue that is mentioned e.g. by [7] is the need for building “pattern languages”, i.e. a collection of architectural styles, for different domains in addition to identifying generic architectural styles that can be used over a number of domains. In this article we outline a path forward in identifying essentials of MAS architectures for different application domains, i.e. MAS architectural styles, and collecting these in an evaluation framework. It is our hope that using this framework will enable more well-informed architectural decisions when creating MAS architectures and MAS systems.

Naturally, there are limitations to the proposed evaluation method. Firstly, it only evaluates the *potential* of different architectural styles. A good implementation may achieve this potential, and a bad implementation may not reach the potential at all. When developing a software system, the potential of the chosen architecture is one important influence of the resulting system, but there are others. For example, familiarity with a particular architectural style, development organization, and coding standards may also influence the final result. Secondly, which architecture candidate the evaluation framework proposes is strongly dependent on the priorities of the quality attributes and the desired load that is fed into the framework. Hence, care must be taken when prioritizing the needs of the system so that the priorities are in fact truly representing the needs for the target system. Thirdly, the quantitative suggestion that the framework produces should be seen as one input among many to the decision process. Other inputs may include e.g. previous experiences or intuition.

In this study we simulate the candidate architectural styles for a particular application domain and instrument the AHP decision support hierarchy with measurements gathered from this simulation. A potential shortcoming of this approach is that we are using simulated data and therefore may get simulated results. However, this is often the best we can do, as it may be very expensive, and sometimes even impossible, to actually fully implement all the candidate architectures and measure the performance

of the deployed system. Although we use simulated data, we argue that the suggested approach is a step forward compared to the *ad hoc* and subjective choice between candidate architectures that is currently often the case.

To gather empirical data from simulation experiments is possible for some quality attributes that have easily defined metrics. However, there are many quality attributes that are not as easily measured. For example, in Section 1.2 we list the attributes Robustness and Modifiability as being of interest in a MAS setting. For these quality attributes a possible extension of the evaluation method outlined in this article would be to make use of AHP's ability to deal with a mixture of tangible data and subjective judgments (successfully used in other studies, e.g. [21]).

4 Conclusions and future work

We describe an approach based on AHP for multi-criteria evaluation of different MAS architectural styles. It is applied to dynamic resource allocation in an experimental study of implementations of instantiations of four architectural styles for load balancing and overload control in Intelligent Networks.

Previous work indicated that asynchronous architectures react faster than synchronous, and that centralized asynchronous architectures better utilize the available resources, although having larger delays and consume more bandwidth when the load is high [11]. With the proposed use of AHP, however, we are not only able to test hypothesis as the ones we just described. We are also able to:

- Quantify the differences in goodness of the candidate architectures according to the desired balance between quality attributes.
- Weight the different scenarios continuously.
- Easily add new instrumentations to increase the granularity of the evaluation.
- Easily add new architectures to evaluate.
- Easily add new evaluation criteria.

The results of the case study are, not very surprisingly, that different architectures excel in different dimensions. The choice of MAS architecture for a particular application should hence be based on a trade-off between the dimensions (e.g. the involved quality attributes) 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 the practice. Our plans for future work include:

- Experimental validation in another dynamic resource allocation domain.
- Use the approach to investigate other domains than dynamic resource allocation.
- Further develop the application of the AHP method in MAS settings, e.g. to include qualitative measures of factors such as robustness and maintainability.
- Investigating to what extent the implementations of the individual agents influence system performance.
- Further development of the concept of architectural styles for characterizing MAS.

Acknowledgements

The authors would like to thank Blekinge Institute of Technology for funding this work.

References

1. D. R. Anderson, D. J. Sweeney, and T. A. Williams. *An Introduction to Management Science: Quantitative Approaches to Decision Making*. South Western College Publishing, Cincinnati Ohio, 2000.
2. A. 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.
3. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Publishing Co., Reading MA, 1998.
4. A. 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.
5. 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.
6. J. Bosch. *Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach*. Addison-Wesley, Harlow UK, 2000.
7. F. Buschmann, C. Jäkel, R. Meunier, H. Rohnert, and M. Stahl. *Pattern-Oriented Software Architecture - A System of Patterns*. John Wiley, Chichester UK, 1996.
8. S. Clearwater, editor. *Market-Based Control: Some early lessons*. World Scientific, 1996.
9. P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. Addison Wesley, 2002.
10. P. R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge MA, 1995.
11. S. 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.
12. J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, 1997.
13. J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14-15):938–947, 1998.
14. P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, pages 42–50, July 1995.
15. T. Magedanz and R. Popescu-Zeletin. *Intelligent Networks*. International Thomson Computer Press, 1996.
16. J. McCall. *Encyclopedia of Software Engineering*, chapter Quality Factors, pages 959–969. John Wiley & Sons Inc., 1994.
17. T. L. Saaty. *The Analytic Hierarchy Process*. McGraw Hill, Inc., New York NY, 1980.
18. T. L. Saaty and L. G. Vargas. *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Kluwer Academic Publisher, Dordrecht the Netherlands, 2001.
19. M. Shaw and D. Garlan. *Software Architecture - Perspectives on an Emergin Discipline*. Prentice Hall, Upper Saddle River NJ, 1996.
20. M. Shepperd, S. Barker, and M. Aylett. The analytic hierarchy process and almost dataless prediction. In R. J. Kuster, A. Cowderoy, F. Heemstra, and E. P. van Veenendaal, editors, *Project Control for Software Quality - Proceedings of ESCOM-SCOPE 99*, Maastricht the Netherlands, 1999. Shaker Publishing BV.
21. M. Svahnberg. An industrial study on building consensus around software architectures and quality attributes. *Journal of Information and Software Technology*, 46(12):805–818, 2004.
22. F. Ygge. *Market-Oriented Programming and its Application to Power Load Management*. PhD thesis, Lund University, Sweden, 1998.