Characterization and Evaluation of Multi-Agent System Architectural Styles

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. We argue that it is useful to study classes of Multi-Agent System (MAS) architectures, corresponding to architectural styles in addition to particular architectures. In this work we focus on a particular abstraction level where MAS architectural styles are characterized according to properties, such as, the type of control used (from fully centralized to fully distributed), and the type of coordination used. Different architectural styles support different quality attributes to different extent. When choosing architectural style for a given application domain, we argue that it is important to evaluate the them according to the quality attributes relevant to that application. The architectural style that provides the most appropriate balance between these attributes should then be selected. As a case study we investigate the problem of dynamic and distributed resource allocation and compare six MAS architectural styles that can be used to handle this task. We also illustrate the use of 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, for selecting the architectural style that balance the trade-off between the relevant quality attributes in the best way.

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 may be characterized and evaluated in a more general way. Typically, 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, without drawing any general conclusions. We believe that this area has now reached the level of maturity when it is appropriate to compare and evaluate MAS architectures on a more abstract level. In this paper, we show how the concept of *architectural styles* can be used to achieve this. We will also illustrate how to choose the proper architectural style 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 architectural style 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 as a case study investigate the problem of selecting a MAS architectural style for dynamic and distributed resource allocation.

The rest of the paper is structured as follows. In Section 2 we will discuss Architectural styles followed by a case study in Section 3. Section 4 discusses the method and finally in Section 5 we draw a few conclusions and suggest some future directions of research.

2 Architectural Styles

As mentioned earlier, we argue that it is useful to study classes of MAS architectures, corresponding to *architectural styles* [18] 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 [13].

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 via hierarchical to fully distributed), and the type of coordination (synchronous vs. asynchronous). The degree of *synchronization* is a way of characterizing the coordination in terms of how the execution of the agents interrelate with each other. We may have agents that are highly sophisticated, but who only interact 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*. As well as having an intermediate level of centralization, we may study architectural styles that exhibit properties in between being fully synchronous and totally asynchronous. However, we limit this work to $\{$synchronous, asynchronous$\}\times\{$centralized, hierarchical, distributed$\}$ architectural styles.

It should be noted that the terminology varies between different sources. Shaw & Garlan [18] introduced the concept of *architectural styles*. In their work, an architectural style consists of components, connectors, and constraints, and defines a family of systems with a specific pattern of structural composition. This encompassed higher level architectural styles such as *client-server, pipes and filters, repositories/blackboard,* and *layered*, but also lower levels such as *object-oriented, dataflow,* and *event-based*. Buschmann et al [5] presents a taxonomy containing, among others, *pipes and filters, blackboard,* and *layered*, and presents these as *architectural patterns*. We can thus interpret architectural patterns to be a subset of Shaw & Garlan's architectural styles. Bosch [4] uses the term *architectural style* in the same way as Shaw & Garlan, but also uses the term *architectural pattern* to denote a lower level solution that can be merged with an architectural style, such as *concurrency, persistence, distribution,* and *graphical user interface*. In this article we use the term architectural style in the same meaning as Shaw & Garlan [18]. In other words, we see an architectural style as an abstraction over a family of systems. Thus, an architectural style is used as a starting point when creating a concrete architecture for a particular MAS system.

3 Case Study

We will now illustrate the use of our method by going through a case study, starting with a description of the domain and the chosen quality attributes. We then present the six MAS architectural styles[1] and their qualitative as well as quantitative evaluations.

3.1 Domain

Since agent technology has shown to be successful for *dynamic and distributed resource allocation*, e.g. power load management [22], cellular phone bandwidth allocation [3], and transportation systems management [9, 21], we have chosen this domain for our architectural style comparison. Basically, the problem concerns allocation of resources between a number of *customers*, given a number of *providers*. Both the providers and the customers may reside at different geographical locations, hence the distributed aspect of the problem. 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.

3.2 Quality Attributes

It is possible to evaluate MAS architectural styles with respect to several different quality attributes [7]. 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 and distributed 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?*
- 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 [14] that could be addressed,[2]

[1] $\{\text{synchronous, asynchronous}\} \times \{\text{centralized, hierarchical, distributed}\}$.

[2] In further addition to these attributes, there are of course a number of attributes that are not mentioned here but that are of interest to include in the specific evaluation of the specific case. The method as such does of course not exclude any quality attributes. It is up to the architectural style evaluator to set the specific attributes when applying the method.

– 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)?*

It is impossible to find a MAS architectural style that is optimal with respect to all the attributes relevant for a certain application. Instead, there is typically a trade-off between these attributes and different architectures balance this trade-off in various ways. Different applications, on the other hand, often require different balancing of this trade-off. Thus, in order to choose the right architecture for a particular application, knowledge about relevant attributes and how different MAS architectural styles support them is essential.

3.3 Candidate MAS Architectural Styles

There are many ways of characterizing the space of possible MAS architectures, e.g., the topology 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 and will discuss and compare the following six potential MAS architectural styles for dynamic and distributed resource allocation:

– centralized synchronous architectures,
– centralized asynchronous architectures,
– hierarchical synchronous architectures,
– hierarchical asynchronous architectures,
– distributed synchronous architectures, and
– distributed asynchronous architectures.

3.4 Qualitative Evaluation

We will now briefly discuss how the architectural style may influence the quality attributes identified above.

– *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.
– The ability to *utilize* the *resources* seems to be favored by centralized, asynchronous solutions which improve the fairness (and thus the utilization) in near-overload situations, and may have better reactivity (which leads to better utilization of resources in highly dynamic domains).

- 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.
- In a distributed system, the reallocation may function partially even though some agents have failed, although the probability of failures in *one* of these controler nodes is higher than the probability of failure in the single node of the centralized solution. At this level of abstraction, it is hard to see that the *robustness* is clearly favored by any of the two properties.
- *The modifiability,* to add or remove a provider or customer, may be slightly better in centralized architectures since changes may only be necessary in one part of the system. However, this attribute seems to be more dependent on other architectural style properties not considered in this evaluation.
- *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.

It is important to note that this analysis can only say something about the *potential* of a particular architectural style. Thus, there is no guarantee that an implemented instantiation of a architectural style actually realizes the potential even though some support for the claims above may be found in previous work where we simulated four instantiations of the architectural styles mentioned [10] which were later evaluated using AHP [8].

3.5 Quantitative 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 architectural style for a particular application knowledge about relevant attributes and how different MAS architectural styles support them is essential. We will now show how the trade-off between quality attributes can be quantified.

The Analytic Hierarchy Process (AHP) [16, 17] is a multi-criteria decision support method from Management Science [1] that has previously been successfully tried and used in software engineering settings similar to the use in this article (e.g. [11, 12, 19, 20]). 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 measured data e.g. from a simulation.

The following steps are identified in our method:

Property	React.	Load Bal.	Fairness	Utiliz.	Respons.	Com. OH	Robustness	Modifi.	Scala.
Priority P_c	0.10	0.05	0.10	0.05	0.10	0.30	0.10	0.10	0.10
Priority P_u	0.10	0.20	0.10	0.20	0.10	0.00	0.10	0.10	0.10

Table 1. Priorities of the various properties in the case of a restricted communication (P_c) and limited resources (P_u).

- 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, the top-level goal is *Most Appropriate Architectural Style*. Under this root node in the hierarchy the different evaluation criteria are listed, in our case *Reactivity*, *Load Balancing*, *Fairness*, *Utilization of Resources*, *Responsiveness*, *Communication Overhead*, *Robustness*, *Modifiability*, and *Scalability*. For some of these criteria a further specialization may be necessary, e.g., the expected load of the target system. In that case, sub-criteria should be added as children in the hierarchy.
- For a particular application, the criteria are then prioritized in accordance with how important they are for that application. This prioritization is done for all levels in the hierarchy, 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. If the quality attributes are not independent, care may be taken when setting the weights of the dependent attributes so that their interaction does not lead to unexpected effects in the evaluation.

  As an illustration, we provide two examples of priorities for the different quality attributes shown in Table 1. These two cases corresponds to one situation where the (potential) system bottleneck lies in the communication network (P_c) and one where the resources are the limiting factor (P_u). In the first situation it is important to keep communication overhead at a low level, whereas in the second situation it is not. Instead, utilization of the resources and load balancing are prioritized.

  We include the two different priorities 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. They can be derived in a multitude of ways (for a comparison of different prioritization techniques, see e.g. [12]), including AHP, and the 100-point method.
- For each of the leaf nodes in the decision support hierarchy we compare each of the candidate architectural styles with the other candidates. This can be done by using a pair-wise comparison process or by providing tangible data. In this study, we use the subjective judgments that are presented in Table 2.
- The obtained normalized values for the candidate architectural styles are then multiplied with the normalized priorities for each level in the decision support hierarchy.

– 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 not absolute numbers but a ratio compared to the other alternatives that is obtained.

  Thus, using the data 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. For each of the two cases we take the product of the priorities of the quality attributes, and multiply this with the corresponding value for each candidate architectural style. The result of this is then summed for each candidate architectural style, and presented in Table 3. As can be seen, in the first case P_c, with restricted communication abilities, the distributed asynchronous architectural style is the most suitable, followed by the other two asynchronous styles. In the second case P_u, with restricted computing resources, the centralized architectural styles seems to be the best choice and the asynchronous version the first option.

4 Discussion

Naturally, there are limitations to the suggested 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.

	Synchronous			Asynchronous		
	Centralized	Hierarchical	Distributed	Centralized	Hierarchical	Distributed
Reactivity	0.033	0.033	0.033	0.300	0.300	0.300
Load Balancing	0.300	0.150	0.050	0.300	0.150	0.050
Fairness	0.300	0.150	0.050	0.300	0.150	0.050
Util. of resources	0.167	0.167	0.167	0.167	0.167	0.167
Responsiveness	0.167	0.167	0.167	0.167	0.167	0.167
Com. overhead	0.080	0.120	0.200	0.120	0.180	0.300
Robustness	0.050	0.150	0.300	0.050	0.150	0.300
Modifiability	0.300	0.150	0.050	0.300	0.150	0.050
Scalability	0.050	0.150	0.300	0.050	0.150	0.300

Table 2. The score of each of the properties of the six architectural styles.

P_c	
Distributed asynchronous	0.218
Hierarchical asynchronous	0.177
Centralized asynchronous	0.176
Distributed synchronous	0.161
Centralized synchronous	0.137
Hierarchical synchronous	0.132

P_u	
Centralized asynchronous	0.210
Centralized synchronous	0.183
Hierarchical asynchronous	0.170
Distributed asynchronous	0.160
Hierarchical synchronous	0.143
Distributed synchronous	0.133

Table 3. Results of the AHP given the two priorities P_c and P_u.

Secondly, which architecture candidate the evaluation framework proposes is highly dependent on the priorities of the quality attributes and the way we choose to define them. 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.

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 in concrete domains. In a previous study [8], we investigated the problem of load balancing and overload control of Intelligent Networks, a dynamic and distributed resource allocation problem. Four concrete MAS architectures were instantiated corresponding to four different architectural styles (centralized synchronous, centralized asynchronous, hierarchical synchronous, and distributed asynchronous). Metrics were defined for six different quality attributes (Reactivity, Load balancing, Fairness, Utilization of resources, Responsiveness, and Communication overhead). The instantiations were studied in simulation experiments and measurements of the metrics were recorded. The measurements where then used as raw data for the AHP in a similar way as the subjective judgments were used in this article. Moreover, this work concerning architectural styles and their implementation 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. [5].

It can of course be questioned if this is a purely objective method. In one way that is an important discussion, however somewhat irrelevant since the method itself does not define the quality attributes. Instead the method should be seen as a structured tool that may be used by the system designer to choose an appropriate architectural style based on his/her definitions and weights of the quality attributes. In fact, the suggested method isolates and makes explicit the subjective parts of the evaluation, i.e., the priorities and the scores, and separates them from the objective parts, i.e., the AHP calculations.

Finally, the work presented in this paper can be viewed in the perspective of a more general evaluation framework, wich 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 quality attributes.

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. We argue that this work will contribute to bridge the current gap between theory and application of MAS.

5 Conclusions and Future Work

Architectural styles have received considerable attention in the software engineering community during the past 10 years (cf. ref. [2, 5, 4, 18]) 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 Buschmann et al. [5] 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 have outlined a path forward in identifying essentials of MAS architectures for different application domains, i.e. MAS architectural styles. We also described a way of evaluating such styles according multiple criteria for different applications and situations. The method was applied to the problem of dynamic and distributed resource allocation where six different MAS architectural styles were evaluated according to nine different quality attributes in two different situations (priorities between the attributes).

The results of the case study are, not very surprisingly, that different architectural styles excel in different situations. The choice of MAS architectural style for a particular application should hence be based on a trade-off between 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:

- Further development of the concept of architectural styles for characterizing multi-agent systems.
- Further experimental validation in dynamic distribution resource allocation domains.
- Investigate the applicability of the suggested evaluation method in other domains.
- Investigating to what extent the implementations of the individual agents influence system performance.
- Compare the approach to other methods of selecting architectural styles, e.g. Qualitative evaluation [6] and ELECTRE [15].

Acknowledgements

The authors would like to thank Blekinge Institute of Technology and the Swedish Knowledge Foundation 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. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Publishing Co., Reading MA, 1998.
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. J. Bosch. *Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach*. Addison-Wesley, Harlow UK, 2000.
5. F. Buschmann, C. Jäkel, R. Meunier, H. Rohnert, and M. Stahl. *Pattern-Oriented Software Architecture - A System of Patterns*. John Wiley, Chichester UK, 1996.
6. L. Chung, K. Cooper, and A. Yi. Developing adaptable software architectures using design patterns: an nfr approach. *Comput. Stand. Interfaces*, 25(3):253–260, 2003.
7. P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. Addison Wesley, 2002.
8. P. Davidsson, S. Johansson, and M. Svahnberg. Characterization and evaluation of multi-agent system architectural styles. In *Software Engineering for Multi-Agent Systems IV*, Lecture Notes in Computer Science. Springer Verlag, 2006. To appear.
9. J. Himoff, P. Skobelev, and M. Wooldridge. Magenta technology: Multi-agent systems for industrial logistics. In *Proceedings of Autonomous Agents and Multi Agent Systems*, volume Industry Track, pages 60–66. ACM press, 2005.
10. S. Johansson, P. Davidsson, and M. Kristell. Four architectures for dynamic resource allocation. In A. Karmouch, T. Magedanz, and J. Delgado, editors, *Mobile Agents for Telecommunication Applications*, volume 2521 of *LNAI*, pages 239–248. Springer Verlag, 2002.
11. J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, 1997.
12. 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.
13. P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, pages 42–50, July 1995.
14. J. McCall. *Encyclopedia of Software Engineering*, chapter Quality Factors, pages 959–969. John Wiley & Sons Inc., 1994.
15. J. C. McPhail and D. Deugo. Deciding on a pattern. In *IEA/AIE '01: Proceedings of the 14th International conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 901–910, London, UK, 2001. Springer-Verlag.
16. T. L. Saaty. *The Analytic Hierarchy Process*. McGraw Hill, Inc., New York NY, 1980.
17. T. L. Saaty and L. G. Vargas. *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Kluwer Academic Publisher, Dordrecht the Netherlands, 2001.
18. M. Shaw and D. Garlan. *Software Architecture - Perspectives on an Emergin Discipline*. Prentice Hall, Upper Saddle River NJ, 1996.

19. 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.
20. 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.
21. D. Weyns, K. Schelfthout, T. Holvoet, and T. Lefever. Decentralized control of E'GV transportation systems. In *Proceedings of Autonomous Agents and Multi Agent Systems*, volume Industry Track, pages 67–74. ACM press, 2005.
22. F. Ygge. *Market-Oriented Programming and its Application to Power Load Management*. PhD thesis, Lund University, Sweden, 1998.