First Workshop on Sharing and Reusing Architectural Knowledge

Lago, Patricia; Avgeriou, Paris

Published in:
EPRINTS-BOOK-TITLE

Publication date:
2006

- Contacting any of the organizers and authors of the SELMAS papers for more information.

Finally, a high-quality set of workshop and invited papers is going to appear in the fifth edition of the book Software Engineering for Multi-Agent Systems (LNCS, Springer, 2007). In addition, this book will include several invited papers. The SELMAS 2007 workshop is planned for the next year at ICSE 2007, and we look forward to an excellent program also in the next year.

Acknowledgements

The organizers would like to thank all those who contributed with submissions to the workshop and the program committee members who invested many hours reviewing such submissions. In addition, we thank the session chairs for the fine work in coordinating the sessions and the keynote speakers for the interesting talks. Finally, we would sincerely like to thank again the SELMAS 2006 participants for their active involvement in the meeting and the level of their contributions to the debate.

References

[1] Choren, R., A. Garcia, H. Giese, H-f. Leung, C. Lucena, A. Romanovsky (Eds.). Proceedings of the 5th Workshop on Software Engineering for Large-Scale Multi-Agent Systems. International Conference on Software Engineering (ICSE 2006), Shanghai, China, 2006.

[2] Choren, R., A. Garcia, C. Lucena, A. Romanovsky (2005). Software Engineering for Multi-Agent Systems III, Research Issues and Practical Applications. Lecture Notes in Computer Science, State-of-the-Art Survey, Vol. 3390, Springer, 2005.

[3] Cristian, F. A Recovery Mechanism for Modular Software. In: Proceedings of the Fourth International Conference on Software Engineering (ICSE 1979), p. 42-50, 1979.

[4] Garcia, A., R. Choren, C. Lucena, P. Giorgini, T. Holvoet, A. Romanovsky (2006): Software Engineering for Multi-Agent Systems IV, Research Issues and Practical Applications. Lecture Notes in Computer Science, State-of-the-Art Survey, Vol. 3914, Springer, 2006.

[5] Garcia, A., C. Lucena, J. Castro, F. Zambonelli, A. Omicini (2003): Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications. Lecture Notes in Computer Science, State-of-the-Art Survey, Vol. 2603, Springer, 2003.

[6] Lucena, C., A. Garcia, A. Romanovsky, J. Castro, and P. Alencar (2004): Software Engineering for Multi-Agent Systems II, Research Issues and Practical Applications. Lecture Notes in Computer Science, State-of-the-Art Survey, Vol. 2940, Springer, 2004.

[7] Parnas, D., H. Würges. Response to Undesired Events in Software Systems. In: Proceedings of the Second International Conference on Software engineering (ICSE 1976), p. 437-446, 1976.

First Workshop on Sharing and Reusing Architectural Knowledge

Patricia Lago
Vrije Universiteit
Amsterdam, The Netherlands
patricia@cs.vu.n

Paris Avgeriou
University of Groningen
Groningen, the Netherlands
<paris@cs.rug.nl>

Abstract

The first SHARK (SHAring and Reusing architectural Knowledge) workshop, attempted to explore the state of the art as well as the state of the practice in this emerging field. This workshop report presents the themes of the workshop, it summarizes the results of the discussions held about various topics, and suggests some research topics that are worthwhile to pursue in the future.

Keywords

Architectural knowledge, Architectural decisions, Software Architecture, Software Reuse.

1. INTRODUCTION

Part or most of software systems is provided through COTS components, outsourcing, open source, multi-party collaboration and distributed development teams. Software architecture plays an increasingly important role to manage the complex interactions and dependencies between the stakeholders and to provide a central artifact that can be used for reference by them. Existing notational and documentation approaches to software architecture typically focus on the components and connectors and fail to document the design decisions that resulted in the architecture as well as the organizational, process and business rationale underlying the design decisions. This results in high maintenance cost, high degrees of design erosion and lack of information and documentation of relevant architectural knowledge.

The workshop focuses on current approaches, tackling this problem: methods, languages, notations, tools to extract, represent, share, use and re-use architectural knowledge. Architectural Knowledge (AK) is defined as the integrated representation of the software architecture of a software-intensive system (or a family of systems), the architectural design decisions, and the external context/environment.

The theme of sharing and reusing architectural knowledge is complex and multi-faceted, both in its core and in its relevance to other advances of software engineering. Overall, it involves at least the following topics:

- Notations to model architectural knowledge
- Ontologies, domain models and meta-models for architectural knowledge
- Communicating, sharing and using architectural knowledge
- Case studies for sharing and reusing architectural knowledge
- Tools to extract, represent, share or use architectural knowledge
- Knowledge grids for sharing architectural knowledge
- Methods and tools to master the evolution of architectural knowledge

- Software patterns as a form of architectural knowledge
- Sharing architectural knowledge in the context of service-oriented architectures (SOA) or Model-Driven Engineering (MDE)
- Communicating architectural knowledge in open, inner and private communities
- Traceability between requirements, architectural design decisions and architectural models

The workshop was organized in two parts: in the morning session, the accepted papers were presented, followed by short discussions regarding each paper; in the afternoon session, first a set of discussion topics was defined among the participants and afterwards two parallel discussion groups elaborated on those topics.

The rest of this workshop report is organized as follows: Section 2 describes the results of the discussion that took place during the two parallel discussion groups and the conclusions reached by the participants. Section 3 outlines the publication plans for the research and position papers, presented in the workshop, while Section 4 concludes with a brief epilogue.

2. DISCUSSION OUTCOME

The presentation of the position and technical papers and the subsequent discussion for each paper formed a set of essential topics for further dialogue among the workshop participants. These topics can be grouped into two categories:

- Architectural Knowledge as a **product**, which concerns architectural knowledge *per se*, and includes the following topics:
  - Architectural design decisions, architectural solutions (e.g. patterns, tactics, reference architectures) and the mutual influence between decisions, solutions and the system quality attributes.
  - Meta-models, domain models, and conceptual models of architectural knowledge.
  - Notations, languages and views for describing and documenting architectural knowledge.
- Architectural Knowledge as a **process**, which deals with *using* architectural knowledge during the software development lifecycle, and includes the following topics:
  - Use cases (or goals) of architectural knowledge.
  - Tools, services, and application-generic infrastructure for supporting the use of architectural knowledge.
  - Methods for discovery and reuse of architectural knowledge.
  - Mechanisms combining architectural knowledge with other fields/disciplines (e.g. variability, aspect-oriented paradigm, versioning).

It is noted that these two categories have overlapping themes which makes it rather difficult to separate them in a discussion. For example a language must be supported by a tool, or the concepts defined in a meta-model must participate in use cases.

The workshop participants split into two discussion groups, each one dealing with one of the above two groups of topics. The discussions within the two groups are summarized in the following subsections.

2.1 Architectural Knowledge as a product

The discussion begun with the topic of architectural decisions, architectural solutions and the relationship between both of them and the quality attributes. First, we argued that the requirements, both functional and non-functional in the form of quality attributes must be explicit and linked to the architectural decisions. The mapping between requirements and decisions is not 1-to-1 but N-to-M. Second, the mapping between the decisions and the architectural solutions, also has an N-to-M cardinality. Figure 1 depicts these relations between the sets of requirements, decisions and solutions. Even though each decision is the selection of one among several alternative solutions, this is not depicted in Figure 1 in order to keep it simple. In theory, the traceability between these three sets would be highly desirable, as the bridge between requirements and architectural solutions would be established. In practice, this traceability is difficult due to the lack of knowledge on how to explicitly and unambiguously relate the sets, as well as the effort required to do so. The way the requirements, solutions and quality attributes can be related, could potentially be resolved by introducing architectural viewpoints and views (views and viewpoints are defined in [2]).

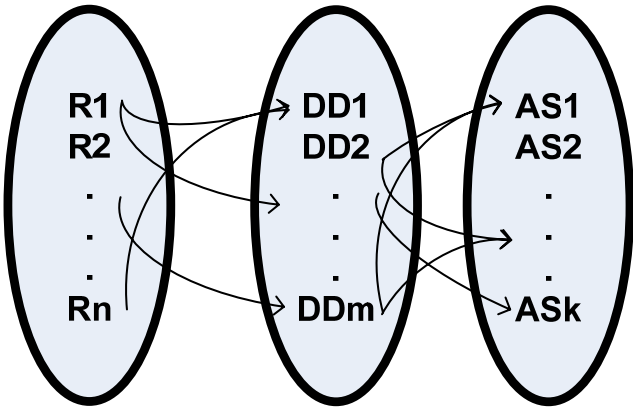

Figure 1 – Mapping between requirements, Architectural Decisions and Architectural Solutions.

The topic of architectural views that describe architectural decisions was debated between the participants. The approach by Capilla et al. [1], that was presented earlier in the workshop was the starting point for relating decisions with requirements and architectural solutions through the concept of views. Specifically, the participants argued that architectural design decisions are horizontal and cross-cutting through the software architecture and thus should be described in a view of their own rather than be scattered in other views. This view can be called "Decision View" and is seen as complementary to the Use Case view as described in [4], which transcends the rest of the architectural views. The Decision View is related to the Use Case view or any other view that describes the requirements: every decision should in fact satisfy partially or wholly one or more requirements. This is in accord to the aforementioned relationship of requirements and architectural design decisions as an N-to-M relationship(see Figure 1). However, the participants stressed the importance of having not only functional requirements (as in use cases) but also quality attributes related to architectural design decisions. A rather large impediment is foreseen in this approach: it is difficult to provide a notation or language in order to realize the mapping between the views. There is a gap between the semantics of requirements, architectural design decisions and architectural solutions, as they come from different paradigms and abide to different metamodels. This is of course not a problem specific to design decisions, but a general problem in software engineering.

The discussion group also addressed the potential tool support in depicting views as well as the relation and traceability between requirements, design decisions and architectural solutions. Since in practice, architectural knowledge is seldom documented, tool support is viewed as the only means to encourage this documentation systematically, if possible to semi-automate it. The size and complexity of software development projects render tool support mandatory rather than desirable. The traceability issue is also relevant here, as an appropriate tool must not only be able to

document architectural design decisions but also to link them to both the requirements and the architectural solutions. Another important feature that would bring added value to the software architect, would be to provide a "playback" of the decisions taken and their impact on the architectural solutions and the quality attributes. In this way, the architect can have an overview of the architecting process and thus be able to backtrack or modify the architecture during the development lifecycle.

The discussion group went on to talk about metamodels, and conceptual models of architectural knowledge. Though much discussion on architectural decisions and knowledge can be found in the literature (for example see [3][5][6]), we first needed to clarify the distinction between two different types of knowledge in order to avoid the ambiguity that is usually associated with this term:

- The **application-generic** knowledge, that architects have implicitly in their heads, from their former experience in working in one or more domains. It is a form of a "library" of knowledge, which consists of e.g. architectural patterns, tactics or reference architectures or even other software engineering techniques, e.g. in requirements engineering. This type of knowledge can be generally applied in several applications independently of the domain.
- The **application-specific** knowledge, of a specific application during the initial development or evolution of that application. This involves all the decisions that were taken during the architecting process of a particular system and the architectural solutions that implemented the decisions.

These two types of knowledge are related in the sense that application-generic knowledge is used in order to take decisions for a single application and thus construct application-specific knowledge. We could simply claim that reusing application-generic knowledge results in the creation of application-specific knowledge.

We then adopted the position that software architecting can be regarded as a decision-making activity: architects consider a number of alternative solutions that could solve the problem statement, and select the one that is deemed as the optimal. We proposed to take this one step further, and consider this decision-making process as a problem-solving activity: the stakeholders define the problem to be solved and the architect selects the optimal solution from a set of alternative solutions.
Figure 2 gives an overview of this process.

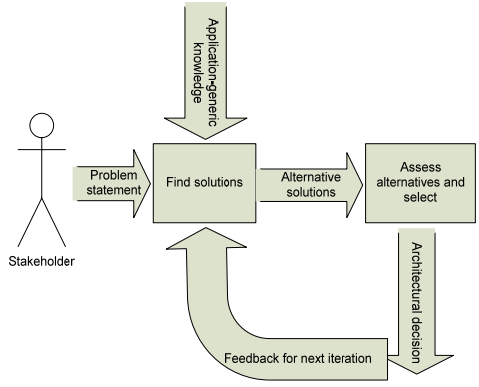

Figure 2 – The architecting process as a problem-solving activity

We propose the following elements as the core concepts in the problem statement:

- The functional requirements and the non-functional requirements, or quality attributes.
- The domain or business model that sets the space where the application exists in.
- The constraints that shape the selection of the solution (e.g. organizational constraints such as processes and resources)
- The technologies, practices or standards that are important and can be used in this application.

All the elements of the problem statement are set by one or more of the stakeholders, including the architect who is a special kind of a stakeholder.

The architect then takes the problem statement as input and tries to identify a set of alternative solutions that could solve it. In order to achieve that, the architect uses the application-generic knowledge as discussed above, for example architectural patterns, architectural tactics or reference architectures. Each alternative solution that the architect finds is accompanied with at least a *rationale*, as well as both positive and negative *consequences* on the system and its quality attributes. The architect subsequently chooses one of the alternative solutions that optimally addresses the problem statement. This choice is in fact an architectural decision and it is a result of a tradeoff among the alternative solutions.

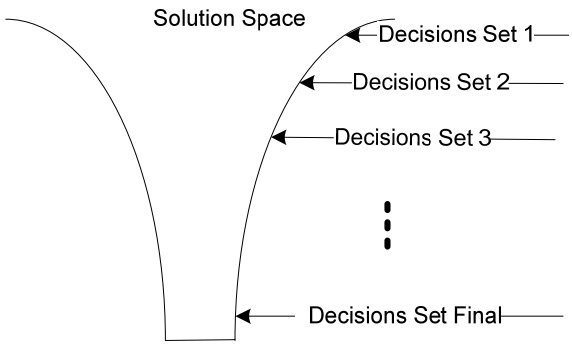

Figure 3 – The architectural design funnel

The solutions chosen, or in other words the decisions taken, are subsequently used through a feedback loop for the next iteration of the architecting process until the architecture of the system is considered stable. The iterative nature of the decision-making process is depicted in Figure 3. The solution space is in the beginning wide and unconstrained and is gradually constrained while the architect finalizes more and more of the architectural decisions. In each decisions set, the existing decisions may be optimized and more decisions may be added. Eventually, the solution space is constrained to a minimum by the final decisions set, that constitutes the application-specific knowledge.

The discussion group further clarified that in practice, architects find only one solution and not multiple alternatives to choose from. This is due to the hard constraints in industrial practice (e.g. time to market or budget) that forces architects to intuitively come up with a single solution based on their existing application-generic knowledge. In effect, this results in the architects not exploring the solution space and potentially missing optimal solutions.

2.2 Architectural Knowledge as a process

As a start the group tried to get a common understanding of the four subtopics around AK usage. As a first result, all the topics have been better defined and refined (see Figure 4). From their experience, the participants identified two types of use cases: *basic use cases*, which are general purpose and provide the building blocks to define special purpose use cases; and *composite use cases* which correspond to special purpose use cases aiming at the achievement of a well defined result. The second topic (tools, services and infrastructure) identifies technologies supporting AK

usage in general. They 'implement' use cases, as well as methods and mechanisms to their realization. The term 'method' (corresponding to the third topic) has been defined as a number of process steps, and the term 'mechanism" (fourth topic) as smaller means or technical solutions to be used in combination with others.

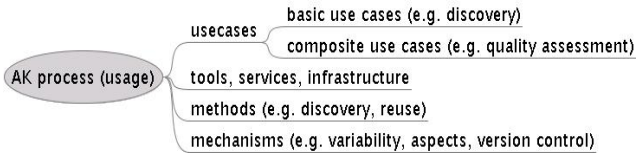

Figure 4 – Topics in Architectural Knowledge usage

The second result of this discussion around terminology is that we could not completely decouple the four topics: composite use cases are always explained in terms of basic use cases, tools always aid a certain (set of) composite use cases, mechanisms vary depending on tools or composite use cases, and methods support a specific AK usage. We think that AK usage can be explained in terms of the four topics, and that they together can characterize AK usage from multiple perspectives. Therefore we used a table to identify first the list of *composite use cases* in using AK, and then for each composite use case define the required elements for each of the other topics. We obtained a table with the following columns:

[basic use cases]-[composite use cases]-[tools]-[meth]-[mech]

The discussion led to four major composite use cases:

- AK quality assessment (i.e. architecture assessments based on AK usage),
- learning AK, both positive and negative (like bad decisions or inconsistencies),
- architecting (defined as creating and checking AK),
- sharing AK (defined as making AK available to others).

We agreed that these composite use cases seem exhaustive in defining AK usage.

We could make the following observations:

1. **Some direct relations cannot be drawn.** We are not always able to draw the correspondence between topics not placed in adjacent columns. For example, "learning AK" (composite use case) can be composed of at least "browsing" and "navigation" (composite use case), and learning can be achieved by "training" (method). If we consider the method "training" we are not able to link it directly in terms of "browsing" and "navigation": to do that we need to use the composite use case "learning AK" as intermediate key reading concept. This suggests a kind of hierarchy in the topics.

2. **It is difficult to draw boundaries.** Almost all participants used or are using a use case driven approach in researching AK usage. This is very intuitive, and it is also effective in industrial investigations and experimentations. Nonetheless, it is less effective if we aim at drawing the boundaries between e.g. composite use cases. For example, architecting AK and sharing AK could be supported by the same tools, and it is difficult to decouple the features aimed at one or at the other. After some discussion we opted for a *consumer/producer* approach as drawn in Figure 5. In this way we naturally split the composite use cases among the two types of stakeholders: producers architect AK and thereafter make it available; consumers use the available AK to either learn from it or carry out some quality assessment.

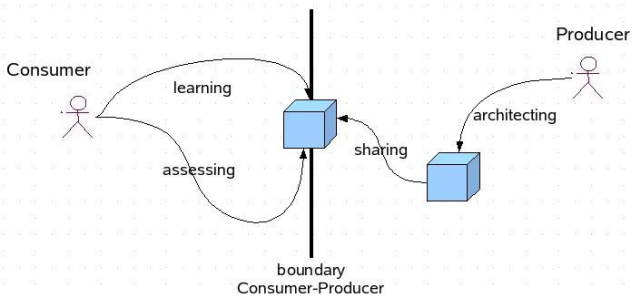

Figure 5 - Boundary between AK producers and consumers

A third concluding observation is the following: the four topics (or aspects) described above are rather straightforward but still they seem very effective in defining AK usage. The workshop working sessions have been too short to be able to discuss extensively the various topics for all composite use cases. In any case, we think that this represents a pragmatic way to rationalize about the use cases and their building blocks with the aim to define what is required whenever AK usage must be engineered. The complete table has been distributed to all workshop participants. Future work is needed in this respect. Possible collaborations have been also discussed.

3. PUBLICATION ACTIVITIES

The papers accepted and presented in the workshop are available on the ACM Digital Library. The paper abstracts and this report are also being published in the ACM SIGSOFT Software Engineering Notes. Some of the workshop papers will be also considered for an extended version to be submitted to the section on Software Architecture of the Journal of Systems and Software.

4. EPILOGUE

In addition to the presented papers and the discussion they raised, the workshop resulted in deeper insights about the open research issues in sharing and reusing architectural knowledge. This report gives a summary of these insights, and a starting point for future research efforts. The workshop has verified the increasing interest on architectural knowledge within the software engineering and architecture community. It also helped to strengthen the links in this small but growing community. The successful output of this workshop has encouraged the organizers to continue with a sequel in the near future.

5. ACKNOWLEDGMENTS

This workshop is part of the dissemination activities of the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge.

We extend our thanks to all those who have participated in the organization of this workshop, particularly to the program committee, which is comprised of: Jan Bosch, Nokia Research Center, Finland, Rafael Capilla, Universidad Rey Juan Carlos, Spain, Torgeir Dingsoyr, Sintef, Trondheim, Norway, Dieter Hammer, University of Groningen, The Netherlands. Paola Inverardi, University of L'Aquila, Italy, Philippe Kruchten, University of British Columbia, Canada, Eltjo Poort, LogicaCMG, The Netherlands, Antony Tang, Swinburne University of Technology, Australia, Hans van Vliet, Vrije Universiteit, The Netherlands, Uwe Zdun, Vienna University of Economics, Austria

We also wish to thank the ACM SIGSOFT for granting the in-cooperation support.

6. REFERENCES

[1] R. Capilla, F. Nava, S. Pérez, J.C. Duenas. A Web-based Tool for Managing Architectural Design Decisions, *Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge*, co-located at the International Conference on Software Reuse, 11-15 June, 2006, Torino, Italy.

[2] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, *IEEE Std. 1471-2000* (2000).

[3] A. G. J. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of WICSA 5*, pages 109–119, November 2005.

[4] P. Kruchten. Architectural Blueprints. The "4+1" View Model of Software Architecture, *IEEE Software 12 (6)*, pp.42-50 (1995).

[5] P. Kruchten. An ontology of architectural design decisions in software intensive systems. In *2nd Groningen Workshop on Software Variability*, pages 54–61, December 2004.

[6] J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Software*, 22(2):19–27, 2005

Report on the International Workshop on Service Oriented Software Engineering (IW-SOSE06)

Elisabetta Di Nitto
Politecnico di Milano
Milano, Italy
dinitto@elet.polimi.it

Robert J. Hall
AT&T Labs Research
Florham Park, USA
hall@research.att.com

Jun Han
Swinburne University of Technology
Melbourne, Australia
jhan@ict.swin.edu.au

Yanbo Han
Chinese Academy of Sciences
Beijing, China
yhan@ict.ac.cn

Andrea Polini
ISTI/CNR
Pisa, Italy
andrea.polini@isti.cnr.it

Kurt Sandkuhl
Jönköping University
Jönköping, Sweden
Kurt.Sandkuhl@ing.hj.se

Andrea Zisman
City University
London, UK
a.zisman@soi.city.ac.uk

Abstract
This paper presents a report of the International Workshop on Service Oriented Software Engineering colocated with ICSE2006. In particular, we shortly present the papers that have been accepted for publication in the workshop proceedings, the keynote speech, and the discussion topics that have emerged during the workshop.

Introduction
Software engineering practitioners and researchers continue to face huge challenges in the development, maintenance, and use of software systems. This has been even more prominent with the new paradigm of service oriented computing in which service integrators, developers, and providers need to create methods, tools, and techniques to support cost-effective development and use of dependable services and service oriented applications. From a technological point of view, recent years have seen the emergence of important standards enabling the Service Oriented vision. However, the engineering of complex and dependable service oriented software still lacks powerful, effective methods and tools.

The International Workshop on Service Oriented Software Engineering (IW-SOSE'06) took place on the the 27th and 28th of May in Shanghai, China, together with the International Conference on Software Engineering (ICSE 2006). The workshop focused on the presentations and discussions of a wide range of topics related to the new paradigm of service oriented software engineering and brought together researchers and practitioners working in the areas of software system engineering and service-oriented.

The two-days workshop included (i) one invited key-note presentation, (ii) presentations of 13 papers rigorously selected by the Programme Committee, and (iii) open round table discussions of the topics of the various papers presented in the workshop. The 13 papers included in the programme of the workshop have been selected from a total of 27 submitted papers. Each submitted paper was reviewed by three members of the Programme Committee.

The papers included in the programme of the workshop represent both industrial and academic perspectives of service oriented software engineering and generated interesting discussions. The papers have been organised into five sessions covering a wide range of issues related to (a) service composition, (b) performance of service oriented systems, (c) service description, (d) service discovery and binding, and (e) service oriented system modelling and application. We present below a brief description of the various sessions together with the titles, author names, and abstract of each paper in the order that they were presented in the workshop and a summary of the discussions.

Program

Keynote Session – Supporting the Composition of Distributed Business Processes
The keynote speaker was Paolo Traverso, head of division at ITC/IRST, Trento, Italy. His main research interests are in automated planning, knowledge representation, meta-level reasoning, formal verification, and automated composition of web services. In the following we report the abstract of his talk.