Proceedings of the 1st International Workshops From Requirements to Architecture (STRAW), co-located with ICSE 2001, Toronto, Canada, May 2001, pp. 42-47

Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach

Alexander Egyed
Teknowledge Corporation
4640 Admiralty Way, Suite 231
Marina Del Rey, CA 90232, USA
+1 310 578 5350
aegyed@acm.org

Paul Grünbacher
Johannes Kepler University
Systems Engineering and Automation
4040 Linz, Austria
+43 732 2468 8867
pg@sea.uni-linz.ac.at

Nenad Medvidovic
University of Southern California
941 W. 37th Place
Los Angeles, CA 90089-0781, USA
+1 213 740 5579
neno@usc.edu

Abstract

Though acknowledged as being very closely related, requirements engineering and architecture modeling have been pursued largely independently of one another in the past years. The inter-dependencies and constraints between architectural elements and requirements elements are thus not well-understood and subsequently only little guidance is available in bridging requirements and architectures. This paper identifies a number of relevant relationships we have identified in the process of trying to relate a requirements engineering approach with an architecture-centered approach. Our approach, called CBSP (Component-Bus-System, and Properties) provides an intermediate language for representing requirements in an architectural fashion. In this paper, we will present the basics of our CBSP approach but also emphasize the challenges that still need to be resolved.

1. Introduction

Requirements largely describe aspects of the problem to be solved and constraints on the solution. Requirements are derived from the problem domain (e.g., medical informatics, E-commerce, avionics, mobile robotics) and reflect the, sometimes conflicting, interests of a given set of system's stakeholders (customers, users, managers, developers). Requirements deal with concepts such as goals, conflicts (issues), alternatives (options), agreements, [3], and, above all, desired system features and properties (both functional and non-functional).

Architectures, on the other hand, model a solution to the problem described in the requirements. Software architectures provide high-level abstractions for representing the structure, behavior, and key properties of a software system. The terminology and concepts used to describe architectures differ from those used for the requirements. An architecture deals with components, which are the computational and data elements in a software system [10]. The interactions among components are captured within explicit software connectors (or buses) [11]. Components and connectors are composed into specific software system topologies. And, architectures capture and reflect the key desired properties of the system under construction (e.g., reliability, performance, cost) [11].

The relationship between a set of requirements and an effective architecture for a desired system, however, is not readily obvious. This conflicts our need of having requirements engineering and architectural modeling being intertwined and mutually-dependent development activities in order to ensure their complete and consistent treatment (i.e., refinement). In context of requirements, architectural modeling has to satisfy the roles of (1) supporting fast trade-off analyses about requirements' feasibility via the modeling of architectural options, and (2) supporting the modeling of architectural solutions in a manner that reflects functional and non-functional properties of requirements in a form that is more readily refineable to code. In context of architectural modeling, requirements engineering has to define (1) functional and non-functional constraints that affect architectural decisions, and, (2) rationale that defines purpose and goal of architectural solutions.

The existence of conceptual differences between what to do (requirements) versus how to do it (architecture, design, and code) constitutes a gap. This gap has been often observed and frequently documented [9], however, despite massive attention it remains unsolved. It is still a difficult problem on how to transition from requirements to an architecture and vice versa. Some of those many issues involve: How to interpret informal requirements in context of more formal architectures? How to elicit functional and non-functional aspects out of requirements? How to infer architectural topologies and styles out of constraints imposed by given requirements? How to reason about mismatches among requirements or between requirements and given architectural solutions? How to maintain requirements and architectures interpedently and yet consistently while both are being evolved? And how to do all of the above if hundreds if not thousands of requirements need to be considered?

To address these challenges and others we have developed a light-weight method for identifying key architectural elements and their dependencies based on given requirements. Our method, called the CBSP approach (Component-Bus-System-Property), helps in refining a set of given requirements into potential architectures by applying a taxonomy of architectural dimensions. Input to our method can be a set of (incomplete) requirements captured in textual or formal descriptions and containing rationale. The result of CBSP is an intermediate model that captures "architectural decisions" of requirements in form of an incomplete architecture.

At the current state we applied CBSP in context of EasyWinWin [2,6], a requirements elicitation technique, and C2 [12], an architectural style for highly-distributed systems. However, we believe that it can be applied to other requirements elicitation and architecture-capture approaches. The following discusses the basics of our CBSP approach in context of an example followed by an update of the current state of the approach and needed future work to make our technique more comprehensive.

2. Cargo Router Case Study

We have performed a thorough requirements, architecture, and design modeling exercise to evaluate CBSP in the context of a cargo router application. The *Cargo Route*r system was built to handle the delivery of cargo from delivery ports (e.g., shipping docks or airports) to warehouses. Cargo is moved via vehicles (e.g., trucks and trains) which are selected based on terrain, weather, accessibility and other factors. The primarily responsibility of the system's user is to initiate and monitor the routing of cargo through a GUI. The user can also request reports and estimations on cargo arrival times and vehicle status (e.g., idle, in use, under repair).

We used the EasyWinWin tool to gather and negotiate requirements for the cargo router system. The WinWin negotiation model [4] and its supporting tool (EasyWinWin [2]) are based on four artifact types: Win Conditions, Issues, Options and Agreements. Win conditions capture the stakeholder goals and concerns with respect to the new system. If a Win condition is non-controversial, it is adopted by an Agreement. Otherwise, an Issue artifact is created to record the resulting conflict among Win Conditions. Options allow stakeholders to suggest alternative solutions, which address Issues. Finally Agreements may be used to adopt an Option, which resolves the Issue.

Three stakeholders participated in a 1-hour brainstorming session and gathered 81 statements (stakeholder win conditions) about its goals.

3. CBSP Steps

To create a "CBSP view" of a given set of requirements, we identified a five-step process [7], four of which are tool supported by EasyWinWin. In this section we will discuss all five steps in context of the Cargo router example.

Identify Core Requirements

Our approach is meant to be used in an iterative manner, where requirements get continuously added or changed. Thus, initially, we found it useful and necessary to reduce the complexity of a given problem by identifying core requirements. In this step, stakeholders vote about importance and relevance of a requirement. Naturally, requirements that did not get included in this step can be included in a future iteration of our process.

Architectural Classification of Requirements

To identify architecture-relevant information out of the pool of requirements, we used a voting process to categorize requirements into six CBSP dimensions (C,B,S,CP,BP,SP). We thus asked all stakeholders to individually decide whether they believe the given requirements could contain component-relevant information (C), bus (connector)-relevant information (B), or is a more general system requirement (S) that affects a larger part of the architecture. Since we were also interested in non-functional requirements, we also gave the stakeholders the option to vote for C-B-S properties (CP, BP, and SP). For instance, the requirement "R09: Support cargo arrival and vehicle availability estimation" was voted to be strongly component-relevant by all stakeholders whereas the requirement "R25: the system must be operational within 18 months" was not voted to be architecturally relevant. Some requirements also received contradictory votes: The requirement "R10: Automatic routing of vehicles" was voted component[1] relevant (C) by all stakeholders but only system property (SP) relevant by one stakeholder.

Identification and resolution of mismatches

As the last example showed, stakeholders may have distinct interpretations of requirements. Naturally, those discrepancies may lead to distinct interpretations of the architectural relevance of those requirements. Indeed, these are the kinds of conflicts we are seeking since the mapping from requirements to architecture often is a matter of understanding the meaning of requirements in context of architectures. The conflict above indicates a case where two stakeholders have different opinions. A subsequent discussion step thus has to be initiated to identify and

[1] Components can be either data components or processing components but that discussion is out of the scope here.

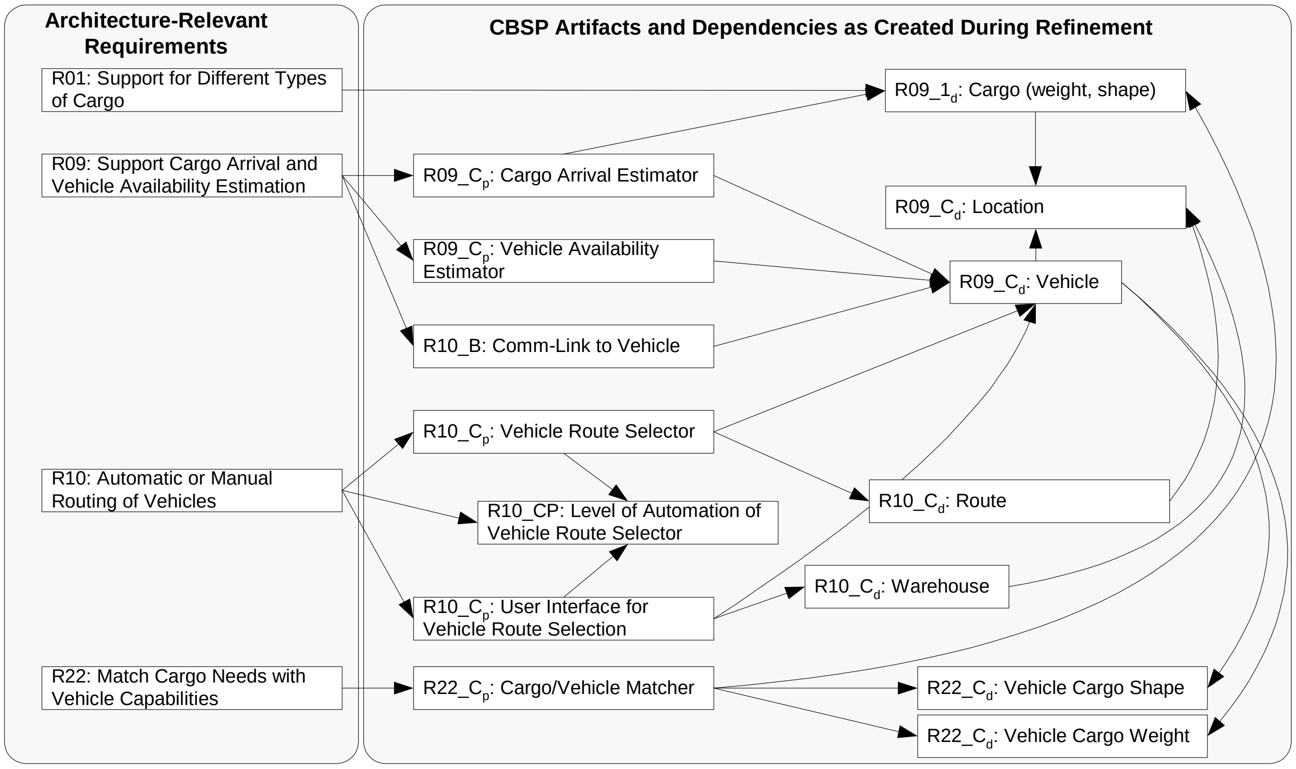
Architecture-Relevant Requirements
CBSP Artifacts and Dependencies as Created During Refinement
R01: Support for Different Types of Cargo
R09: Support Cargo Arrival and Vehicle Availability Estimation
R10: Automatic or Manual Routing of Vehicles
R22: Match Cargo Needs with Vehicle Capabilities
R09_1d: Cargo (weight, shape)
R09_Cp: Cargo Arrival Estimator
R09_Cd: Location
R09_Cp: Vehicle Availability Estimator
R09_Cd: Vehicle
R10_B: Comm-Link to Vehicle
R10_Cp: Vehicle Route Selector
R10_Cd: Route
R10_CP: Level of Automation of Vehicle Route Selector
R10_Cd: Warehouse
R10_Cp: User Interface for Vehicle Route Selection
R22_Cp: Cargo/Vehicle Matcher
R22_Cd: Vehicle Cargo Shape
R22_Cd: Vehicle Cargo Weight

Figure 1: Sample Artifact Relationships in Cargo Router Example.

resolve that difference. In above example, one stakeholder thought this requirement implied that (1) the system needs to suggest paths that vehicles travel (e.g., via navigation points) but not their sources and targets whereas another stakeholder thought this requirement implied that (2) the system also needs to identify the sources and destinations for vehicles. The discussion thus clarified this conflict and an instant re-vote identified this requirement as component relevant and system property relevant (SP).

Table 1: Concordance Matrix.

Consensus	ACCEPT requirement as architecturally relevant if at least one *largely* or *fully* vote	REJECT requirement as not architecturally relevant if no vote higher than *partially*
Conflict	DISCUSS and resolve reason of conflict before proceeding (e.g., properties are implicitly captured and often ambiguous)	

Table 1 shows rules that describe conflict handling during CBSP voting. In case of consensus among the stakeholders, the requirements are either accepted or rejected based on the voted degree of architectural relevance (note that stakeholders are given the option of four votes: no, partial, strong, or full architectural relevance; our tool provides statistical reasoning on how to infer consensus). If the stakeholders cannot agree on the relevance of a requirement to the architecture, they further discuss it to reveal the reasons for the different opinions until a point of consensus is reached. This typically leads to a clarification of the particular requirements as above example has shown. Figure 1 (left) depicts a few requirements that were voted to be architecturally relevant.

Architectural refinement of requirements

Architecturally relevant requirements explicate at least one CBSP dimension that all stakeholders agreed on to be relevant. Obviously, some requirements address multiple dimensions. For instance, the requirement "R09: Support cargo arrival and vehicle availability estimation" was voted to be fully component relevant, fully system relevant, and largely bus relevant. In order to understand requirements better and to better relate them to other requirements it is necessary to refine them into more atomic entities.

CBSP dimensions also play an important role in the refinement process of requirements. For instance, the requirement "R09: support cargo arrival and vehicle availability estimation" was determined C, B, and S relevant. This implied that the refinement should reveal component, bus, and system relevant aspects. Indeed, on a high level, this requirement supports two processing components: "R09_C_p Cargo arrival estimator" and "R09_C_p Vehicle availability estimator." Cargo arrival estimator depends on data components like cargo ("R09_C_d Cargo"), the vehicle ("R09_ C_d Vehicle") it is on and the location of the vehicle ("R09_ C_d Location"). Vehicle availability estimator only depends on the knowledge of the vehicle and its location (but not cargo). Above requirement was also rated bus-relevant. This was the case because the location of a vehicle (and its cargo) is variable as it moves. A connector (bus) is therefore needed to allow the system to track vehicles (see Figure 1 right). Note that we now do not talk about requirements any more but instead of pieces of architecturally-relevant information elicited from requirements – we refer to those pieces as CBSP artifacts.

It must be noted at this point, that generally only component, bus, and system artifacts are seen as candidates for refinement. Properties (CP, BP, SP) are harder to refine since they tend to span large parts of a system.

Derivation of Architectural Style and Architecture

CBSP artifacts and their dependencies are valuable for architectural modeling. For instance, we can see that the estimator components depend on vehicle information, thus, indicating potential architectural implications. Naturally, CBSP artifacts and their dependencies also make dependencies between requirements more explicit. For instance, we can assume some dependency between "R09: support cargo arrival and vehicle availability" and "R10: automatic or manual routing of vehicles" because they both "share" the CBSP artifact "R09_C_d: Vehicle."

In context of the Cargo Router we found that CBSP artifacts mapped straightforwardly to architectural elements defined in C2. For instance, the C2 architecture has components called *Vehicle* and *Estimator* and the architecture makes use of explicit data connectors (buses) to realize component interactions. However, we are not yet in a position where we could actually derive architectures or styles out of CBSP properties. We discuss this and other issues in future work.

4. Future Work

We found CBSP very useful in organizing requirements and systematically refining them but at the current state some of the activities are still rather labor intensive. Naturally, requirements engineering is people centric, however, this section will discuss how automation can aid stakeholders in coming up with requirements and architectures in a faster more reproducible way. We currently provide tool support for capturing requirements, voting on them, identifying conflicts, refining them, and maintaining trace dependencies. There are, however, a few areas that have not been explored in depth.

Architectural Trade-off Analyses

Thus far we treated requirements and architectures very static and defined how a CBSP approach can bridge the two. However, requirements engineering is much more iterative where stakeholders uncover not just goals but also issues (conflicts) and potential options (solutions). Issues may arise because of architectural conflicts (i.e., no suitable architectural option can be found that satisfies given requirements). We believe that CBSP artifacts are not only useful for refinement but they also provide "feedback loops" in cases where architectural decisions impact requirements. The approach thereby helps to capture findings from architectural modeling and simulation and supports analysis of an architecture for adherence to requirements.

For example, some issues can only be identified after a draft architecture has been modeled and described. These issues and corresponding architectural options can be captured by architects as CBSP elements to capture the rationale of architectural decisions and to relate this rationale with the relevant requirements. (A Bus Property issue (e.g., bottleneck) could be identified through simulation experiments, a component option could be suggested by the architect, etc.) This capturing of tradeoff decisions is similar to the ATAM technique described in [8]

Problems identified through architectural models and simulation can be captured as CBSP elements, such as "I12_S Three seconds system response time not possible." Architectural options and alternative solutions can be also described as CBSP elements. For example "O24_C Consider use of OTS staff management component." CBSP provides as an intermediate model between a requirements and an architecture definition approach that allows "bi-directional" traces; the resulting intermediate model facilitates synthesis of negotiation artifacts into architectural elements and enables feedback from architecture modeling and analysis.

CBSP can also be interpreted as a way to negotiate about architectural concerns (win conditions, issues, options, and agreements) with clearly established links to both the related requirements and the architectural elements.

CBSP Refinement of Issues, Options, and Agreements

If CBSP can be used to diversify requirements engineering via architectural trade-off analyses then naturally the subsequent negotiation results may also be in need of refinement. For instance, if a potential option were suggested to resolve a given issue then it would be desirable if CBSP could support the "explorative" refinement of that option in context of the rest of the system to evaluate its feasibility.

To this end, Figure 2 shows one possible situations where requirements issues and options can be used to guide the refinement process. The left part of the figure shows three win condition (W1, W2, and W3) which could be requirements of an earlier negotiation process or simply new stakeholder goals. The middle part of the figure shows the corresponding CBSP artifacts (note that we did not define their interdependencies in the same detail as we did in Figure 1).

After some time (potentially with the help of architectural modeling) a problem (issue) is encountered between win conditions W1 and W2. It is found that in order to optimize concurrent routings (W1), we need to support bi-directional real-time communication. This contradicts W2 which only requires unidirectional communication from system to vehicle (e.g., to forward routing requests). This conflict could naturally be resolved via an option (O1) that states that we need a two-way bus; an option which would "replace" the win condition W2. In the CBSP view this causes a dilemma in that one requirement "artifact" requires a uni-directional bus whereas the other requires a bi-directional bus. Thus, CBSP refinement would also have to be complemented by a minimization step that would interpret requirements interdependencies (i.e., like the replace link between O1 and W2) to infer needed CBSP artifacts and their interdependencies. We believe that such a minimization step could be largely or even fully automated.

Inconsistency and Incompleteness Issues

Simplifying the refinement of requirements to architecture and reasoning about requirements feasibility in context of architectural modeling are two aspects we believe CBSP could support. However, the relationships between architecture, CBSP, and the negotiation rationale may become very complex when both architecture and requirements evolve independently. To this end, we found that CBSP could also provide powerful support for simplifying inconsistency detection between requirements and architecture. For instance, we observed the following cases:

- Inconsistencies between CBSP artifacts/dependencies and their actual realization: For instance, if a CBSP artifacts is categorized as a component but is implemented as a connector in the architecture then this could indicate a potential lack of understanding of either the requirement or architecture. Similar conclusions can be drawn when CBSP dependencies do not match architectural ones.
- Inconsistencies between architecture/CBSP and negotiation agreements: Take, for instance, the example of the *Optimizer* component in Figure 2 which depends on the *Two-Way-Bus*. If during the WinWin negotiation it is decided to implement the *Optimizer* but at the same it is decided to implement the *One-Way-Bus* instead of the *Two-Way-Bus* (i.e., O1 would not get adopted) then this indicates a potential mismatch (*Optimizer* needs the *Two-Way-Bus*).
- Completeness mismatch between architecture and requirements: For instance, CBSP can help in identifying whether all agreed-upon architecturally-relevant artifacts have actually been realized in the architecture. Similarly, CBSP can help in pointing out if there are any architectural elements for which there are no corresponding negotiation artifacts. Completeness issues such as the ones above could suggest lack of awareness by stakeholders of some architectural aspects that could have influenced the negotiation process and vice versa.

Deriving/Validating Architectural Styles out of CBSP

In our work to date, we have chosen to use architectural styles as guides in transforming the initial architectural decisions produced by CBSP into an actual architecture. Specifically, we have employed the C2 style [12] as discussed above. However, each style is particularly well suited for a certain type of problem; therefore, our intent is to extend CBSP to leverage other styles as well.

We have begun exploring the feasibility of composing CBSP artifacts into an architecture according to the Pipe-and-Filter [11], GenVoca [1], and Weaves [5] styles, in

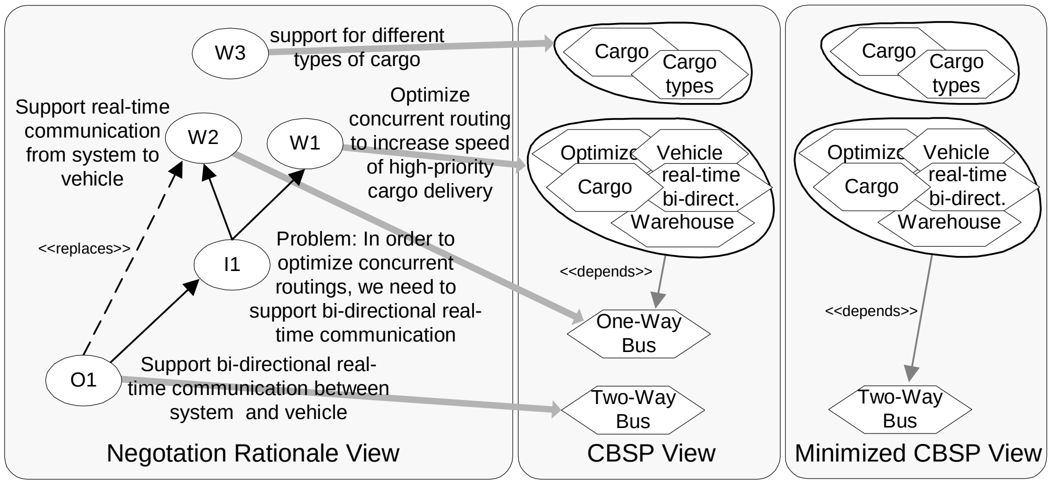
W3
support for different types of cargo
Support real-time communication from system to vehicle
W2
W1
Optimize concurrent routing to increase speed of high-priority cargo delivery
<<replaces>>
I1
Problem: In order to optimize concurrent routings, we need to support bi-directional real-time communication
O1
Support bi-directional real-time communication between system and vehicle
Negotation Rationale View
Cargo
Cargo types
Optimizer
Vehicle
real-time bi-direct.
Warehouse
<<depends>>
One-Way Bus
Two-Way Bus
CBSP View
Minimized CBSP View

Figure 2. From Negotiation rationale to an CBSP view using Minimization

addition to C2. Each style imposes different constraints that guide the composition of CBSP artifacts into an architecture. For example, in the cargo router example, the *Optimizer* CBSP artifact depends on the *Vehicle* and *Warehouse* artifacts. C2's substrate independence principle mandates that *Optimizer* be placed below them in the architecture. Since there are no direct dependencies between *Vehicle* and *Warehouse*, they may be adjacent. The same dependency relationship would have different topological implications in a different style. For example, GenVoca would require *Optimizer* to be above the *Vehicle* and *Warehouse* components (while still allowing *Vehicle* and *Warehouse* to be at the same level). Furthermore, unlike C2, GenVoca would allow direct interactions among its components, without the intervening connectors.

Similarly, if a component in a system, e.g., *Weather Module*, communicates by producing streams of data, while other components, e.g., *Vehicle*, assume discrete event-based communication, then the style selected to represent the architecture must supply explicit software connectors to mediate between the two types of interaction. In this case, GenVoca would not be an adequate candidate, while Weaves, Pipe-and-Filter, and C2 may be, as all three of them provide explicit connectors. However, if we further consider the types of component interaction supported by the three styles, we see that neither C2 nor Pipe-and-Filter provide adequate solutions in this case: C2 assumes purely discrete event-based communication, while Pipe-and-Filter assumes purely data stream-based communication. This would leave Weaves as the obvious choice.

Our future work will center around expanding the number of architectural style we are considering. We will also leverage existing studies on styles to codify the style elimination and selection criteria such as the ones outlined above. Finally, we intend to determine whether there are architectural styles that are inherently incompatible with CBSP, and the reasons for that incompatibility.

5. Conclusions

This paper introduced the CBSP approach for refining requirements to architectures. The process is partially tool supported and is currently integrated with our EasyWinWin negotiation process. Besides requirements refinement, the CBSP process also has great potential for improving a variety of related issues like consistency and conformance and architectural trade-off analyses. Future work involves the exploration of those issues as well as a more tight integration of our models and tools.

Acknowledgements

This work has been supported by the Austrian Science Fund (Erwin Schrödinger Grant 1999/J 1764 "Collaborative Requirements Negotiation Aids").

References

1. Batory D. and O'Malley S.: The Design and Implementation of Hierarchical Software Systems with Reusable Components. ACM Transactions on Software Engineering and Methodology, 1992.
2. Boehm, B. and Gruenbacher, P.: "Supporting Collaborative Requirements Negotiation: The Easy WinWin Approach," Proceedings of SCS Virtual Worlds and Simulation Conference, January 2000.
3. Boehm B., Egyed A., Kwan J., and Madachy R.: Using the WinWin Spiral Model: A Case Study. IEEE Computer, 1998, 33-44.
4. Boehm B. W. and Ross R.: Theory W Software Project Management: Principles and Examples. IEEE Transactions on Software Engineering, 1989, 902-916.
5. Gorlick, M. M. and Razouk, R. R.: "Using Weaves for Software Construction and Analysis," Proceedings fot eh International Conference on Software Engineering (ICSE), Los Alamitos, CA, 1991, pp.23-34.
6. Gruenbacher, P. and Briggs, B.: "Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin," Proceedings of the Hawaii International Conference on Systems Sciences, 2001.
7. Gruenbacher, P., Egyed, A., and Medvidovic, N.: "Reconciling Software Requirements and Architectures: The CBSP Approach," Submitted to International Requirements Engineering Conference (RE), Toronto, Canada.
8. Kazman, R., Barbacci, M., Klein, M., Carričre, S. J., and Woods, S. G.: "Experience with Performing Architecture Tradeoff Analysis," Proceedings of the 21th International Conference on Software Engineering (ICSE), Los Angeles, CA, May 1999, pp.54-63.
9. Medvidovic, N., Gruenbacher, P., Egyed, A., and Boehm, B.: "Software Lifecycle Connectors: Bridging Models across the Lifecycle," submitted to International Conference on Software Engineering and Knowledge Engineering (SEKE 2001), 2001.
10. Perry D. E. and Wolf A. L.: Foundations for the Study of Software Architectures. ACM SIGSOFT Software Engineering Notes, 1992.
11. Shaw, M. and Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
12. Taylor R. N., Medvidovic N., Anderson K. N., Whitehead E. J. Jr., Robbins J. E., Nies K. A., Oreizy P., and Dubrow D. L.: A Component- and Message-Based Architectural Style for GUI Software. IEEE Transactions on Software Engineering 22(6), 1996, 390-406.