Reconciling Software Requirements and Architectures: The CBSP Approach

Paul Grünbacher
Systems Eng. & Automation
Johannes Kepler University
4040 Linz, Austria
pg@sea.uni-linz.ac.at

Alexander Egyed
Teknowledge Corp.
4640 Admiralty Way, Suite 231
Marina Del Rey, CA 90292
aegyed@acm.org

Nenad Medvidovic
Univ. of Southern California
941 W. 37th Place
Los Angeles, CA 90089-0781
neno@usc.edu

Abstract

Little guidance and few methods are available to refine a set of software requirements into an architecture satisfying those requirements. Part of the challenge stems from the fact that requirements and architectures leverage different terms and concepts to capture the artifacts relevant to each. In this paper we will present CBSP, a lightweight approach intended to provide a systematic way of reconciling requirements and architectures. CBSP leverages a simple set of architectural concepts (components, connectors, overall systems, and their properties) to recast the requirements in a way that facilitates their straightforward mapping to architectures. Furthermore, the approach allows us to capture and maintain arbitrarily complex relationships between requirements and architectural artifacts, as well as across different CBSP artifacts. We have extensively applied CBSP within the context of particular requirements and architecture definition techniques, EasyWinWin and C2. We leverage that experience in this paper to demonstrate the CBSP method and tool support using a large-scale example that highlights the transition from an EasyWinWin requirements negotiation into a C2-style architectural model.

1 Introduction

Software systems of today are characterized by increasing size, complexity, distribution, heterogeneity, and lifespan. They demand careful capture and modeling of requirements [20][24] and architectural designs [23][25] early on, before the low-level system details begin to dominate the engineers' attention and significant resources are expended for system construction. Understanding and supporting the interaction between software requirements and architectures remains one of the challenging problems in software engineering research [20]. Evolving and elaborating system requirements into a viable software architecture satisfying those requirements is still a difficult task, mainly based on intuition. Little guidance is available for modeling and understanding the impact of architectural choices on the requirements. Software engineers face some critical challenges when trying to reconcile requirements and architectures:

- Requirements are frequently captured informally in a natural language. On the other hand, entities in a software architecture specification are usually specified in a formal manner [17].
- System properties described in non-functional requirements are commonly hard to specify in an architectural model [17].
- Iterative, concurrent evolution of requirements and architectures demands that the development of an architecture be based on incomplete requirements. Also, certain requirements can only be understood after modeling and even partially implementing the system architecture [7][21].
- Mapping requirements into architectures and maintaining the consistency and traceability between the two is complicated since a single requirement may address multiple architectural concerns and a single architectural element may have numerous non-trivial relations to various requirements.
- Real-world, large-scale systems have to satisfy hundreds, possibly thousands of requirements. It is difficult to identify and refine the architecturally relevant information contained in the requirements due to this scale.
- Requirements and the software architecture emerge in a process involving heterogeneous stakeholders with conflicting goals, expectations, and terminology [3]. Supporting the different stakeholders demands finding the right balance across these divergent interests [13].

To address these challenges we have developed a lightweight method to identify the key architectural elements and the dependencies among those elements, based on the stated system requirements. The CBSP (Component-Bus-System-Property) approach helps to refine a set of requirements into an architecture by applying a taxon-

omy of architectural dimensions. In this iterative process CBSP provides a requirements to architecture model connector that helps to concurrently evolve requirements and architecture models [21]. Input to the method is a set of typically incomplete and often quite general requirements captured as textual descriptions, possibly containing rationale. The result of CBSP is an intermediate model that captures architectural decisions in an incomplete draft architecture; this model guides the selection of a suitable architectural style to be used as a basis for converting the draft architectures into an actual implementation of a software system architecture.

The intent of our work is to provide a generic approach that would work with arbitrary informal or semi-formal requirements elicitation and architecture modeling approaches. In order to validate our research to date, however, we have applied it extensively in the context of EasyWinWin [4][12][13][27], a groupware-supported requirements negotiation approach, and C2 [16], an architectural style for highly distributed systems. Our approach provides:

- a lightweight way of converting requirements into an architecture using a small set of key architectural concepts;
- mechanisms for "pruning" the number of relevant requirements, rendering the technique scalable;
- involvement of key system stakeholders, allowing non-technical personnel (e.g., customers, managers, even users) to influence the system's architecture if desired; and
- adjustable voting mechanisms to resolve conflicts.

Together, these benefits afford a high degree of control over refining large-scale system requirements into architectures.

The paper is organized as follows: Section 2 discusses the details of the CBSP approach. Section 3 describes the application of CBSP to a large-scale example problem. Section 4 describes our current tool support. Related work is discussed in Section 5. Conclusions round out the paper in Section 6.

2 The CBSP Approach

The relationship between a set of requirements and an effective architecture for a desired system is not readily obvious. *Requirements* largely describe aspects of the problem to be solved and constraints on the solution. Requirements are derived from the concepts and relationships in the problem domain (e.g., medical informatics, E-commerce, avionics, mobile robotics). They reflect the, sometimes conflicting, interests of a given system's stakeholders (customers, users, managers, developers). Requirements deal with concepts such as goals, options, agreements, issues, conditions [3], and, above all, desired system features and properties (both functional and non-functional). Requirements may be simple or complex, precise or ambiguous, stated concisely or carefully elaborated. Of particular interest to our work is a large class of requirements that is predominantly stated in a natural language such as English, as opposed to precise formalisms. On the surface, such requirements are easier to understand by humans, but they frequently lead to ambiguity, incompleteness, and inconsistencies.

On the other hand, *architectures* 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 [23]. The interactions among components are captured within explicit software connectors (or buses) [25]. Components and connectors are composed into specific software system topologies. Finally, architectures both capture and reflect the key desired properties of the system under construction (e.g., reliability, performance, cost) [25]. These elements of software architectures are typically specified formally using architecture description languages, or ADLs [17].

The above-described differences between requirements and architectures make it difficult to build a bridge that spans the two. For example, it is unclear in general whether and how a statement of stakeholder goals should affect the desired system's architecture; similarly, deciding how to most effectively address a functional requirement often boils down to relying on the architects' intuition, rather than applying a well-understood methodology. For these reasons, we have formulated CBSP, a technique for relating requirements and architectural models. This technique supports the development of an architecture addressing a given set of requirements in a more straightforward and consistent manner. This section will introduce the CBSP taxonomy of architectural dimensions and describe a process that guides the application of the taxonomy.

2.1 CBSP taxonomy

The fundamental idea behind CBSP is that any software requirement may *explicitly or implicitly* contain information relevant to the software system's architecture. It is frequently very hard to surface this information, as different stakeholders will perceive the same requirement in very different ways [13]. At the same time, this information is often essential, in order to properly understand and satisfy requirements. CBSP supports the task of identifying and elaborating this information. The CBSP dimensions include a set of general architectural concerns

that can be applied to systematically classify and refine requirements artifacts (e.g., specific goals, concerns, options, and so on) and to capture architectural tradeoff issues and options (e.g., impact of connector throughput on the scalability of the topology).

Each requirement is assessed for its relevance to the system architecture's components, connectors (buses), topology of the system or a particular subsystem, and their properties. Thus, each CBSP artifact explicates an architectural concern and represents an early architectural decision for the system. For example, a requirement such as

R: The system should provide an interface to a Web browser.

can be recast into a processing component and a bus CBSP artifact

C_p: A Web browser should be used as a component in the system.

B: A connector should be provided to ensure interoperability with third-party components.

It is important to emphasize that, while CBSP supports recasting requirements into more architecturally "friendly" artifacts along well-defined dimensions, it does not prescribe a particular transformation of a requirement. Instead, our intent is to give a software architect sufficient leeway in selecting the most appropriate *refinement* or, at times, *generalization* of one or more requirements. Examples of both refinement and generalization are given below.

There are six possible CBSP dimensions discussed below and illustrated with a simple example from a spreadsheet manipulation application. The six dimensions involve the basic architectural constructs [17] and, at the same time, reflect the simplicity of the CBSP approach.

1. *C* are artifacts that describe or involve an individual Component in an architecture. For example

   R: Allow user to directly manipulate spreadsheet data.

   may be refined into CBSP artifacts describing both processing components (C_p) and data components(C_d)

   C_p: Spreadsheet manipulation UI component.

   C_d: Data for spreadsheet.

2. *B* are artifacts that describe or imply a Bus (connector). For example

   R: Manipulated spreadsheet data must be stored on the file system.

   may be refined into

   B: Connector enabling interaction between UI and file system components.

3. *S* are artifacts that describe System-wide features or features pertinent to a large subset of the system's components and connectors. For example

   R: The user should be able to select appropriate data filters and visualizations.

   may be refined into

   S: The system should employ a strict separation of data storage, processing, and visualization components.

4. *CP* are artifacts that describe or imply data or processing Component Properties. As discussed above, the properties in CBSP are the "ilities" in a software system, such as reliability, portability, incrementality, scalability, adaptability, and evolvability. For example

   R: The user should be able to visualize the data remotely with minimal perceived latency.

   may be refined into

   CP: The data visualization component should be efficient, supporting incremental updates.

5. *BP* are artifacts that describe or imply Bus Properties. For example

   R: Updates to system functionality should be enabled with minimal downtime.

   may be refined into

   BP: Robust connectors should be provided to facilitate runtime component addition and removal.

6. *SP* are artifacts that describe or imply System (or subsystem) Properties. For example

   R: The spreadsheet data must be encrypted when dispatched across the network.

   may be transformed into

   SP: The system should be secure.

Note that, e.g., the BP example (5.) involved refining a general requirement into a more specific CBSP artifact. On the other hand, the SP example (6.) involved the generalization of a specific requirement into a CBSP artifact.

2.2 CBSP process

In addition to the taxonomy of architectural dimensions, we provide a step-by-step process and techniques supporting the synthesis of the CBSP model and the architecture in a collaborative manner. Figure 1 depicts process activities and deliverables. If used in an iterative process CBSP supports one iteration of evolving and refining an architecture out of a given set of requirements. Each CBSP step is discussed in more detail below.

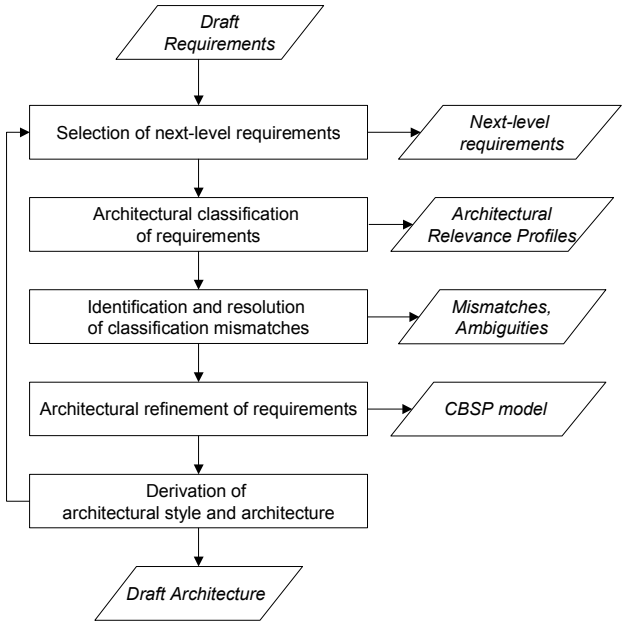

Figure 1: CBSP process.

Selection of next-level requirements. To reduce the complexity of addressing large numbers of requirements, a team of architects applies the CBSP taxonomy to the most essential set of requirements only in each iteration. In this activity we thus eliminate requirements considered unimportant or infeasible through collaborative prioritization, thus arriving at a set of core requirements to be considered for the next level of refinement.

Architectural classification of requirements. A team of *architects* classifies the selected requirements using the CBSP taxonomy. All requirements are assessed by the experts individually on their relevance to the CBSP dimensions using an ordinal scale *(not-partially-largely-fully)*. For instance, a requirement that is rated as *partially* relevant along the component (C) dimension implies that this requirement has some (partial) impact on one or more architectural components. As a result of this step, a profile showing the aggregated architectural relevance (C/B/S/CP/BP/SP) is available for each requirement.

Identification and resolution of classification conflicts. If multiple architects independently perform an architectural classification of requirements using CBSP, their findings may diverge since they may perceive the same statement differently. Revealing the reasons for diverging opinions is an important means of identifying misunderstandings, ambiguous statements, tacit knowledge, and conflicting perceptions [13]. The measured consensus among stakeholders is thus a proxy for their mutual understanding of a requirement's meaning and their agreement on the architectural relevance of a requirement. We determine the level of consensus through a statistical test, Kendall's coefficient of concordance [26].

Table 1 depicts rules that indicate how to proceed in different situations: in case of consensus among stakeholders, the requirements are either accepted or rejected based on the voted degree of architectural relevance. We accept requirement as architecturally relevant if at least one third of the stakeholders vote largely or fully. We reject requirements if no vote is higher than partially. 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. This typically leads to a clarification of a requirement and eases the subsequent step of refining it into one or more architectural dimensions.

Table 1: Concordance/Relevance Matrix.

Concordance \ Relevance	Largely or Fully	Partially or Not
Consensus	**Accept**	**Reject**
Conflict	**Discuss**	

Architectural refinement of requirements. In this activity the team rephrases and splits requirements that exhibit overlapping CBSP properties. Each requirement passing the consensus threshold (concordance and *largely* or *fully* relevant) may need to be refined or rephrased since it may be relevant to several architectural concerns. For instance, if a requirement is *largely* component relevant, *fully* bus relevant, and *largely* bus property relevant, then splitting it up into several architectural decisions using CBSP will increase clarity and precision.

During this process, a given CBSP artifact may appear multiple times as a by-product of different requirements. For example, the following two requirements result in the identification of a *Cargo* data component.

R01: Support for different types of cargo.

R09: Support cargo arrival and vehicle estimation.

Such redundancies are identified and eliminated in the CBSP model. It is also possible to merge multiple related CBSP artifacts and converge on a single artifact.

Derivation of architectural style and architecture. At this point, requirements should have been refined and rephrased into CBSP artifacts in such a manner that no stakeholder conflicts exist and all artifacts are at least largely relevant to one of the six CBSP dimensions. Based on simple CBSP artifacts, a draft-architecture can be derived.

Architectural styles [1][16][23][25] provide rules that exploit recurring structural and interaction patterns across a class of applications and/or domains. Based on the dependencies among the elements in the minimal CBSP view, the rules of the selected style allow us to compose

them into an architecture. In other words, we select the style based on (1) the characteristics of the application domain and (2) the desired properties of the system, identified in the requirements negotiation and elaborated during the CBSP process. This can, of course, result in multiple candidate styles (or no obvious candidates). We choose the best candidate and, based on its rules and heuristics, start converting the CBSP artifacts into components, connectors, configurations, and data, with the desired properties.

3 A Step-By-Step Example

This section illustrates the CBSP approach describing a concrete case study in which we applied the activities described above in the context of the EasyWinWin requirements elicitation method and the C2 architectural style.

Our example application is developed in collaboration with a major U.S. software development organization. It addresses a scenario in which a natural disaster results in extensive material destruction and casualties. In response to the situation, an international humanitarian relief effort is initiated, causing several challenges from a software engineering perspective. These challenges include efficient routing and delivery of large amounts of material aid; wide distribution of participating personnel, equipment, and infrastructure; rapid response to changing circumstances in the field; using existing software for tasks for which it was not intended; and enabling the interoperation of numerous, heterogeneous systems employed by the participating countries. In particular, our system (called *Cargo Router*) must handle the delivery of cargo from delivery ports (e.g., shipping docks or airports) to warehouses close to the distribution centers. Cargo is moved via vehicles (e.g., trucks and trains) selected based on terrain, weather, accessibility and other factors. Our system must also report and estimate cargo arrival times and vehicle status (e.g., idle, in use, under repair). The primarily responsibility of the system's user is to initiate and monitor the routing of cargo through a simple GUI.

We have performed a thorough requirements, architecture, and design modeling exercise to evaluate CBSP in the context of the cargo router application. We used the EasyWinWin method to gather and negotiate requirements for the cargo router system. EasyWinWin is a groupware-supported methodology [4] based on the WinWin approach [3] that aims at enhancing the directness, extent, and frequency of stakeholder interaction. EasyWinWin adopts a set of COTS groupware components (electronic brainstorming, categorizing, voting, etc.) developed at the University of Arizona and commercialized by GroupSystems.com [11].

Three stakeholders participated in a 1-hour brainstorming session and gathered 81 statements (stakeholder win conditions) about their goals. In a first step, the team converged on 64 requirements by reviewing and reconciling similar or redundant goals. For instance, one stakeholder asked to "track location of vehicles" whereas another asked for "the system [to] enable real-time status reports and updates on ports, warehouses, vehicles, and cargo." Obviously, both stakeholders pursued similar goals and, thus, the two win conditions were merged (after verifying stakeholder consensus) into the more general requirement "real-time communication and awareness."

3.1 Selection of next-level requirements

The 64 requirements became the initial baseline for the cargo router project and covered functional and non-functional aspects, software process issues, and time and budget constraints of the system to be developed. To identify the set of requirements needed for a first draft architecture in the first iteration, we performed a joint prioritization of the 64 requirements and selected the 25 requirements with the highest business importance and feasibility.

3.2 Architectural classification of requirements

To extract and surface the architecturally relevant information from the pool of 25 core requirements, a voting process was initiated. Four *architects* classified each requirement with respect to its architectural relevance along the six CBSP dimensions (C/B/S/CP/BP/SP). With four stakeholders involved, a total of 600 votes were cast. This classification process was carried out in less than one hour.

For instance, the requirement "Support cargo arrival and vehicle availability estimation" was voted to be strongly component-relevant by all architects, whereas the requirement "the system must be operational within 18 months" was not voted to be architecturally relevant.[1] Some requirements also received contradictory votes: the requirement "Automatic routing of vehicles" was voted component relevant (C_p and C_d) by all stakeholders, but only system property (SP) relevant by one stakeholder. At the same time, there was a high degree of consensus on other requirements. For example, since stakeholders voted the requirement "support cargo arrival and vehicle availability estimation" to be only component relevant (*largely* and *fully* ratings), this requirement was accepted as architecturally relevant without further discussions.

[1] This does not mean that process and business aspects are, in general, unrelated to a system's architecture. For example, a scheduling conflict may cause architecturally-relevant requirements to be dropped or relaxed.

3.3 Identification and resolution of classification mismatches

Ambiguities in the requirements' meanings led to several conflicts when stakeholders contradicted one another in rating the architectural relevance of requirements. For instance, stakeholders disagreed on the system property (SP) relevance of the requirement "automatic routing of vehicles," and cast ballots for *not*, *partial*, and *full* architectural relevance. The concordance matrix in Table 1 suggests that in such a situation stakeholders need to discuss the mismatch. In this particular case, the discussion revealed different perceptions of what this requirement implied. One stakeholder thought this requirement implied that the system needs to suggest paths that vehicles travel (e.g., via navigation points), but not their sources and targets. Another stakeholder thought this requirement would imply that the system would also need to suggest the sources and destinations for vehicles. The discussion clarified this conflict and an instant re-vote identified this requirement as indeed system property relevant.

From the total of 150 decisions (25 requirements x 6 categories) 43 decisions affecting 19 requirements turned out to be controversial and needed further attention. After all conflicts were resolved, 19 out of the original 25 requirements remained in the pool of architecturally relevant requirements.

3.4 Architectural refinement of requirements

Architecturally relevant requirements explicate at least one CBSP dimension. Some requirements address multiple dimensions. For instance, the requirement "Match cargo needs with vehicle capabilities" was voted to be only processing component (C_p) relevant, whereas the requirement "Support cargo arrival and vehicle availability estimation" was voted to be fully component relevant, fully system relevant, and largely bus relevant. As such, the latter requirement was much more comprehensive than the former. In fact, the latter requirement even depended on the former. For instance, the need for special types of vehicles (e.g., to ship liquid substances) also has an impact on delivery time. In order to better relate such requirements, it is necessary to refine them into more atomic entities. For instance, the fact that cargo arrival estimation depends on vehicle capabilities does not imply that the former requirement fully depends on the latter.

CBSP dimensions also play an important role in the requirements refinement process. For example, the requirement "Match cargo needs with vehicle capabilities" was determined to be only component relevant. As such, the requirement was analyzed and refined into the processing component "Cargo/Vehicle Matcher." This processing component requires as input cargo and vehicle information, resulting in its dependency on relevant data components (e.g., "Cargo weight"). Since some of those data components did not exist beforehand, they were also created. As a result, the refinement of this requirement produced two types of information: (1) CBSP artifacts (processing and components) describing architectural decisions and (2) links describing dependencies among those artifacts. The CBSP artifacts provide potential building blocks for the architecture, whereas the CBSP dependencies help to clarify potential control and data

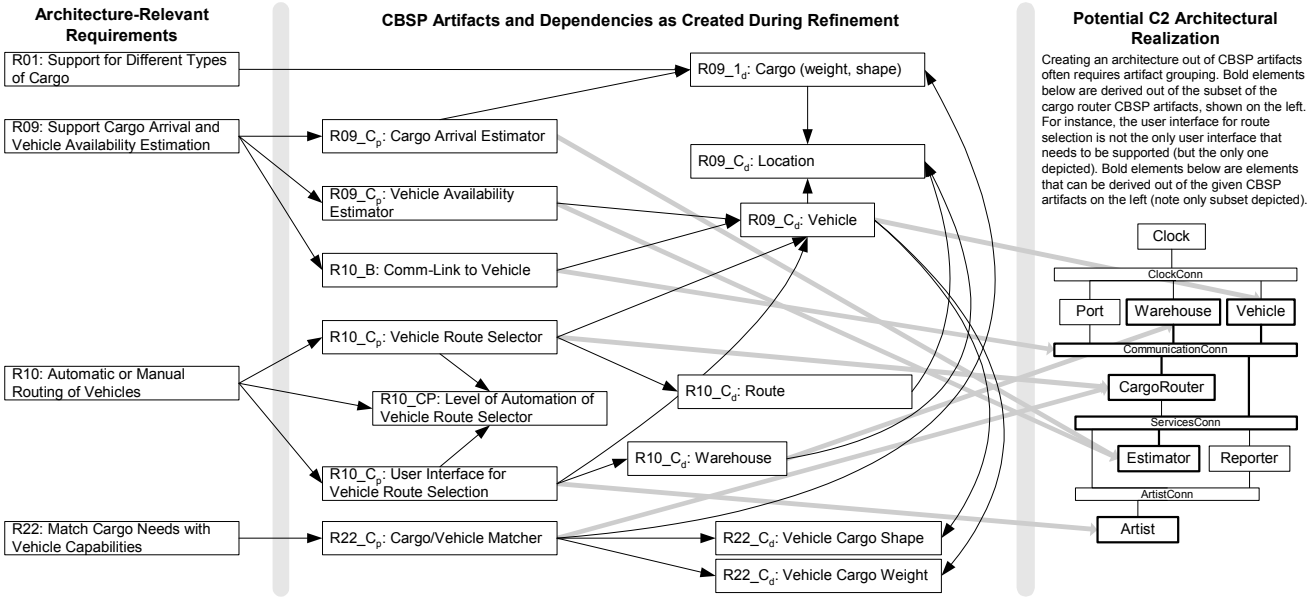

Figure 2: Sample Artifact Relationships in Cargo Router Example.

dependencies within the architecture (which ultimately help architects to find a suitable architectural style).

The refinement process of a more complex requirement is similar but more elaborate. For instance, as discussed above, the requirement "support cargo arrival and vehicle availability estimation" was determined to be C, B, and S relevant. At a high level, this requirement supports two processing components: "Cargo arrival estimator" and "Vehicle availability estimator." Cargo arrival estimator depends on data components representing "Cargo", the "Vehicle" carrying the cargo, and the "Location" of the vehicle. Vehicle availability estimator only depends on the knowledge about the vehicle and its location (but not cargo). The 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 (recall requirement "real-time communication and awareness").

In our experience to date, 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. An example is the requirement "The system should support 30 operators, 15 incoming ports, and 50 warehouses," which was voted system property relevant. From the 25 core requirements, 10 were rated property relevant, while the remaining 15 requirements were refined into CBSP artifacts (e.g., "Vehicle"). In total, 48 CBSP artifacts were created.

Figure 2 shows an excerpt of the CBSP refinement process. The left two columns of the picture depict the refinement of four requirements into CBSP artifacts. As an example, the figure depicts the requirement "Support for different types of cargo" and shows that it was refined into in a simple data component called *Cargo*. The requirement "Support cargo arrival and vehicle availability estimation" was more complex and was broken up into several artifacts including two processing components and a bus component. The figure shows that those artifacts were additionally refined via sub-elements (e.g., vehicle) and dependencies. This additional refinement is not necessary, but can result in useful insights into overall system interdependencies. For instance, we learn that the cargo component is also needed by other requirements, making *Cargo* a centerpiece of the system. Should we later want to remove cargo descriptions from the system, the existing dependencies would also allow us to reason about the impact of this removal on other parts of the architecture and the corresponding requirements. In the context of CBSP, refined artifacts can be merged together resulting in less duplication, but more CBSP artifact dependencies. As stakeholders in our case study refined all requirements into CBSP artifacts, they identified 27 CBSP artifacts across the architecturally relevant requirements.

3.5 Derivation of Architectural Style and Architecture

CBSP artifacts and their dependencies are valuable for architecture and requirements trade-off analyses. They are also useful in creating and modifying architectural representations. For instance, we can see that the estimator components depend on vehicle information, indicating a potential relationship in the architecture. Figure 2 (right column) also shows the refinement of the CBSP artifacts into an architectural realization using the C2 architectural style. Creating an architecture out of CBSP artifacts often requires artifact grouping. Bold elements in Figure 2 are derived out of the subset of the cargo router CBSP artifacts, shown on the left. We find the *Estimator* component placed underneath the *Vehicle* component in the C2 architecture. Recall that C2 topologically requires service-providing components to be put above service-requiring components. The CBSP artifacts (middle column in Figure 2) are either directly represented in C2 or are grouped together into C2 elements (e.g., vehicle data components, estimator processing components). The only CBSP artifact not represented in C2 is *cargo*, which is defined using C2's ADL [16] (not depicted here). The bold items in the C2 architecture in Figure 2 were derived from the depicted CBSP refinement. The remaining items are be derived from other requirements, which have been elided for brevity.

The example in Figure 2 shows one refinement into a C2-style architecture. Naturally, there could be other realizations of the same requirements (in C2 or another architectural style).

4 Tool Support

Our ultimate goal is to provide tool support for each activity in the CBSP process depicted in Figure 1. This section discusses tools we have devised to date for the CBSP method and techniques. To speed up tool development we have adopted off-the-shelf components from GroupSystems.com's groupware infrastructure. Furthermore, we have developed a bridge to the Rational Rose repository to ease transition of requirements into architecture and allow integration of this work with our existing requirements [12], architecture [16], and design [8] tools.

Selection of next-level requirements. This activity is supported as part of the EasyWinWin methodology. Stakeholders use a distributed voting tool to assess requirements for their business importance and feasibility [12].

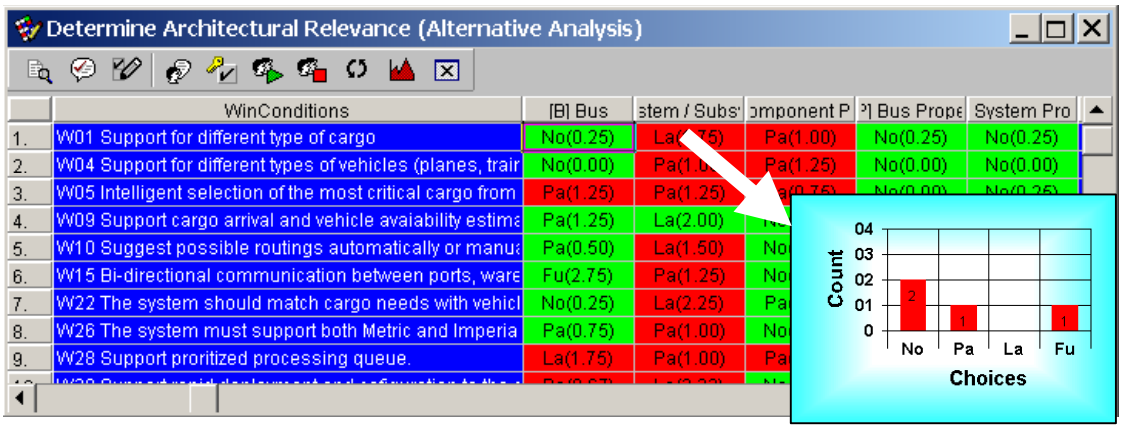

Figure 3: CBSP Classification and Conflict Detection.

Architectural classification of requirements. We customized a voting tool to support the CBSP taxomony. The CBSP dimensions are assessed in a voting process involving multiple experts. Figure 3 depicts voting results (architectural relevance profiles) showing win conditions as *not relevant (No)*, and *partially (Pa)*, *largely (La)*, or *fully (Fu) relevant*.

Identify and resolve classification conflicts. We also provide tool support for identifying and resolving classification mismatches. This is achieved by highlighting conflicting opinions and perceptions. Different cell colors indicate the level of consensus among the experts. Consensus is indicated with green (light gray in Figure 3), while disagreement is indicated with red (dark gray in Figure 3). The vote spread can be displayed (small window in Figure 3) to trigger discussions about differences in opinion.

Architectural refinement of requirements. A prototype interface to the Rational Rose modeling tool is provided to support repository-based integration and refinement of requirements artifacts. The integration from GroupSystems to Rose allows translating WinWin negotiation results and CBSP artifacts into a UML representation. UML stereotypes are used to extend the modeling capabilities and enable artifact types such as win condition, bus property, component, and so on. The Rational Rose repository helps in avoiding and eliminating duplicate items.

Derivation of Architectural Style and Architecture. Although our architectural modeling and analysis tool, DRADEL [16] has been integrated with Rational Rose, we currently do not provide tool support for recommending architectural styles. CBSP artifacts and dependencies do constrain the architectural space in a manner that potentially supports this kind of reasoning. We intend to address this area in our future work.

5 Related Work

The work described in this paper is related to several areas of research covering requirements, architecture, and model transformation.

To date, we have applied CBSP in the context of the WinWin requirements negotiation approach. WinWin is related to other techniques that focus on capturing and evolving stakeholder goals and interests into requirements [6][7][14][18][24]. We believe that CBSP would also work with these approaches.

Refining requirements into architectures is also discussed in the context of processes for requirements capture (e.g., [24]). Our work on refining requirements complements such processes with a structured transformation technique and tool support. However, other approaches that enable automated refinement of requirements (e.g., [19]) are predicated on a more formal treatment of requirements artifacts than a technique such as WinWin would allow.

A key issue in transforming requirements into architecture and further software artifacts is traceability. Researchers have recognized the difficulties in capturing development decisions across modeling artifacts [9]. Gotel and Finkelstein [10] suggest a formal approach for ensuring the traceability of requirements during development. Our approach captures extensive traces thus satisfying many of the needs identified in [9] [10].

Within the area of software architectures, two concepts provide guidance for architects in converting system requirements into effective architectures. The first is architectural styles [25], which captures recurring structural, behavioral, and interaction patterns across applications that are in some way related and/or similar. As discussed above, we indeed make extensive use of architectural styles in formulating an architecture from a collection of CBSP artifacts. The drawback of our current approach is that styles are typically collections of design heuristics,

requiring extensive human involvement and adding a degree of unpredictability to the task of transforming CBSP artifacts into architectures. The second related concept is domain-specific software architecture (DSSA) [28]. A DSSA captures a model of the (well understood) application domain, together with a set of recurring requirements (called reference requirements) and a generic architecture (called reference architecture) common to all applications within the domain. While these DSSA artifacts would make the task of arriving at an architecture from CBSP artifacts even simpler than by leveraging styles, CBSP does not require the existence of a DSSA, nor is it restricted only to extensively studied application domains.

Several approaches have been proposed to ease bridging requirements and architectures. The ATAM technique [14] supports the evaluation of architectures and architectural decision alternatives in light of quality attribute requirements. Nuseibeh [21] describes a twin peaks model that aims at overcoming the often artificial separation of requirements specification and design by intertwining these activities in the software development process. Brandozzi and Perry [5] use the term architecture prescription language for their extension of the KAOS requirements specification language towards architectural dimensions.

Finally, CBSP also relates to the field of transformational programming [2][15][22]. The main differences between transformational programming and CBSP are in their degrees of automation and scale. Transformational programming strives for full automation, though its applicability has been demonstrated primarily on small, well-defined problems [22]. CBSP, on the other hand, can be characterized only as semi-automated; however, we have applied it on larger problems and a more heterogeneous set of models, representative of real development situations.

6 Conclusions and Further Work

We have introduced the CBSP (Component, Bus, System, Property) approach that aims at reconciling software requirements and architectures. We believe that, although a deliberately simple and lightweight approach, CBSP assists in coping with the challenges discussed in the introduction:

- *Bridging different levels of formality:* CBSP provides an intermediate model reducing the semantic gap between high-level requirements and architectural descriptions.
- *Modeling non-functional requirements*: CBSP allows to identify and isolate 'ilities' in requirements at the system level (SP) and architectural-element level (CP, BP), thus improving modeling of non-functional properties.
- *Maintaining evolutionary consistency*: The intermediate model between requirements and architecture produced by CBSP allows specifying more meaningful dependency links that improve evolutionary consistency.
- *Incomplete models and iterative development*: CBSP does not mandate that the requirements be complete. CBSP also allows architects to maintain arbitrarily complex dependencies between a system's requirements and its architecture, thus easing iterations between the two.
- *Handling scale and complexity*: CBSP focuses only on the most essential subset of requirements in each iteration and, further, on the subset of those requirements describing architecturally relevant properties. In fact, each activity in the CBSP process results in filtering out requirements or merging multiple requirements into one. Voting is an important mechanism for reducing complexity by increasing focus and allowing to better understand different stakeholder perceptions.
- *Involving heterogeneous stakeholders*: The semiformal CBSP representation and intuitive tools allow the involvement of success-critical stakeholders in all stages of the process.

In our future work we intend to extend CBSP in the following directions:

- Although demonstrated in the context of EasyWinWin and C2, we believe that CBSP has potential for wide applicability and provides a generic framework of bridging requirements into architecture and design (e.g., UML). Further validation is needed by exploring CBSP using additional requirements and architecture methods.
- Another direction is to devise an approach analyzing the architectural constraints captured as CBSP artifacts/dependencies and recommending architectural styles for a given CBSP model. In particular, the properties described in a CBSP model (e.g., component and bus properties, system wide properties) facilitate the identification of a proper architectural style for a given problem.
- We are also aiming at improving the method to better support capturing feedback from architecture modeling to requirements negotiations, i.e., how findings from architectural modeling, simulation, etc. can be captured as CBSP artifacts (e.g., bus property issue, system property issue, component option, etc.).
- We are interested in extending ADLs (e.g., C2) to better support modeling of properties captured as granular CBSP artifacts.

Acknowledgements

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

This material is also based upon work supported by the National Science Foundation under Grant No. CCR-9985441. Effort also sponsored by the Defense Advanced Research Projects Agency (DARPA), Rome Laboratory, Air Force Materiel Command, USAF under agreement numbers F30602-00-2-0615 and F30602-00-C-0200. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Laboratory or the U.S. Government. Effort also supported in part by Xerox.

References

[1] Batory D., O'Malley S., The Design and Implementation of Hierarchical Software Systems with Reusable Components. ACM Transactions on Software Engineering and Methodology, Oct. 1992.

[2] Bauer F. L., Moller B., Partsch, H., Pepper P. Formal Program Construction by Transformations – Computer-Aided, Intuition-Guided Programming. *IEEE Transactions on Software Engineering*, 15(2), Feb. 1989.

[3] Boehm B., Egyed A., Kwan J., Port D., Shah A., Madachy R., Using the WinWin Spiral Model: A Case Study, IEEE Computer, 7:33–44, 1998.

[4] Boehm B., Grünbacher P., Briggs B., Developing Groupware for Requirements Negotiation: Lessons Learned, IEEE Software, May/June 2001, pp. 46-55.

[5] Brandozzi M., Perry D.E., Transforming Goal-Oriented Requirement Specifications into Architecture Prescriptions, ICSE 2001 Workshop "From Software Requirements to Architectures", 2001.

[6] Dardenne A., Fickas S., van Lamsweerde A., Goal-Directed Concept Acquisition in Requirement Elicitation. *6th Int. Workshop on Software Specification and Design (IWSSD 6)*, Oct. 1993.

[7] Egyed A., Boehm B., Comparing Software System Requirements Negotiation Patterns. *Systems Engineering Journal*, Vol.6/1:1-14, 1999.

[8] Egyed, A. Heterogeneous View Integration and its Automation, Dissertation, Univ. of Southern California, LA, CA 90089-0781, 2000.

[9] Gieszl L. R., Traceability for Integration. *2nd International Conference on Systems Integration (ICSI 92)*, pp. 220-228, 1992.

[10] Gotel O.C.Z., Finkelstein A.C.W., An Analysis of the Requirements Traceability Problem. *1st International Conference on Rqts. Eng.*, pp. 94-101, 1994.

[11] GroupSystems.com, http://www.groupsystems.com

[12] Grünbacher P, Collaborative Requirements Negotiation with EasyWinWin, 2nd International Workshop on the Requirements Engineering Process, Greenwich London, IEEE Computer Society, 2000. pp. 954-960.

[13] Grünbacher P., Briggs B., Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin, Proceedings Hawaii International Conference on System Sciences, IEEE Computer Society, 2001.

[14] Kazman R., Barbacci M., Klein M., Carriere S.J., Woods S.G., Experience with Performing Architecture Tradeoff Analysis, ICSE 99.

[15] J. Liu, Traynor O., Krieg-Bruckner B., Knowledge-Based Transformational Programming. *Fourth International Conference on Software Engineering and Knowledge Engineering*, 1992.

[16] Medvidovic N., Rosenblum D.S., Taylor R.N., A Language and Environment for Architecture-Based Software Development and Evolution, In: Proceedings ICSE'99, pp. 44-53, LA, CA.

[17] Medvidovic N., Taylor R.N., A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, 26/1:70-93, 2000.

[18] Mullery G., CORE: A Method for Controlled Requirements Specification. *ICSE 4*, Munich, Germany, Sept. 1979.

[19] Nuseibeh B., Kramer J., Finkelstein A., A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. *IEEE Transactions on Software Engineering*, Oct. 1994.

[20] Nuseibeh B., S. Easterbrook. Requirements Engineering: A Roadmap, In: The Future of Software Engineering, Special Issue 22nd International Conference on Software Engineering, ACM-IEEE, pp. 37—46, 2000.

[21] Nuseibeh B., Weaving Together Requirements and Architectures. IEEE Computer, 34(3):115-117, March 2001.

[22] Partsch, H., Steinbruggen R., Program Transformation Systems. *ACM Computing Surveys*, 15/3, September 1983.

[23] Perry, D.E., Wolf, A.L., Foundations for the Study of Software Architectures, *Software Engineering Notes*, Oct. 1992.

[24] Robertson, S., Robertson J., *Mastering the Requirements Process*. Addison-Welsey, 1999.

[25] Shaw, M., Garlan D., *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, April 1996.

[26] Siegel S., Castellan NJ. Jr., Nonparametric Statistics for the Behavioral Sciences. New York: McGraw-Hill; 1988.

[27] Stallinger F., Grünbacher P., System Dynamics Modelling and Simulation of Collaborative Requirements Engineering, to appear: Special Issue, Journal of Systems and Software, 2001.

[28] Tracz, W., DSSA (Domain-Specific Software Architecture) Pedagogical Example. ACM SIGSOFT Software Engineering Notes, July 1995.