DOCUMENTING AND ANALYZING A CONTEXT-SENSITIVE DESIGN SPACE

Hans de Bruin, Hans van Vliet, and Ziv Baida
Vrije Universiteit Amsterdam
Mathematics and Computer Science Department
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{hansdb,hans,ziv}@cs.vu.nl

Abstract In most requirements engineering and software architecture documents, emphasis is placed on the chosen alternative. The discarded ones, and the arguments that led to a particular choice, are often not explicitly recorded and documented. This makes it difficult to retrace decisions and explore alternatives. We have developed a representation for capturing quality requirements and associated architectural solution fragments, called the Feature-Solution (FS) graph. We use the knowledge captured in the FS-graph to iteratively compose an architecture. This paper shows that when the knowledge in the FS-graph captures context-sensitive architectural knowledge, such as the concerns of different stakeholders, this representation can also be used to document and reason about architectural trade-offs. The result not only documents feasible architectures, but also the traces of design decisions that led to those architectures, which is a valuable asset during the further implementation and evolution of the system.

1. Introduction

In most requirements engineering documents, emphasis is placed on the chosen alternative. The discarded ones, and the arguments that led to a particular choice, are often not explicitly recorded and documented [17]. This makes it difficult to retrace decisions and explore alternatives. In our experience, the same holds for software architecture documentation.

The architecture of a software system captures early design decisions. These early design decisions reflect major quality concerns, including functionality. In order not to reinvent the wheel time and again, we would like to capture chunks of architectural knowledge explicitly, and use these chunks when deriving an architecture that fulfills some set of quality concerns. Our solution for capturing this knowledge is a Feature-Solution (FS) graph, which connects quality requirements with solution fragments at the architectural level, and allows us to iteratively compose an architecture [8, 9].

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35607-5_15

J. Bosch et al. (eds.), *Software Architecture*

By considering different levels for each, or at least a number of, quality requirements, as well as the associated solution fragments, the FS-graph can be used to explore a design space consisting of architectural solutions for different sets of quality requirements. The FS-graph then documents this design space, and it can be used to explore and analyze the various possible architectural solutions.

Most, if not all, architectural knowledge is context sensitive. It is applicable in certain situations, but not in all. Different stakeholders, for example, have different and often conflicting concerns. Differences in available infrastructure often also result in different architectural solutions. By modeling this contextual variation in the FS-graph as well, we get an even richer picture. To arrive at a particular solution then requires an analysis of the FS-graph to identify and resolve possible conflicts.

We illustrate our use of the FS-graph to model and reason about context-sensitive architectural knowledge with an example from the e-business domain. In the earlier analysis of this case, two feasible solutions were identified [10], each of which is the result of a series of design decisions. By explicitly documenting all context-sensitive architectural knowledge in a FS-graph, a much richer picture is created, which allows us to reason about feasible business models and the concerns they address. Thus used, the FS-graph also provides traceability from requirements and contextual concerns to architectural solutions, which is a valuable asset during the further evolution of the system.

The remainder of this paper is organized as follows. We start with giving a short description of the case we consider in section 2. In section 3, we give background information on the FS-graph. Section 4 shows how the FS-graph can be used to document and analyze the design space for the case. Finally, section 5 discusses related work, and section 6 contains our conclusions.

2. Case Description

We illustrate the use of the FS-graph for capturing and reasoning about architectural solutions by means of the *Amsterdam Times* newspaper case. The case concerns the provisioning of a value-added news service. The newspaper wants to offer its subscribers the ability to read articles online, without additional costs to these customers. The expenses are to be financed by telephone connection revenues, which the reader must pay to set up a telephone connection for Internet connectivity.

The *Amsterdam Times* case is discussed extensively in [10]. A most critical aspect in a business model for this type of application is how value (such as money and content) flows between actors involved. In [10], two different e-business models are discussed: the *terminating* model and the *originating* model. Both models involve several parties: the customer (reader of articles), the newspaper, a local operator, and a telecom operator. In the terminating

model, the reader pays a fee for a telephone connection to the local operator which acts as an intermediary between the customer and *The Amsterdam Times*. The local operator, *The Last Mile*, transfers part of this fee to the newspaper and the telecom operator. In the *originating* model, the reader pays *The Amsterdam Times* directly, and the newspaper then has to pay the other parties involved.

These two business models are the result of a series of design decisions, resulting from choosing specific values for quality requirements. Though the final models as presented in [10] are easy to grasp, the individual decisions which led to these models are much harder to reconstruct. As a result, it is also difficult to see which other solutions are possible. By explicitly documenting all decisions, we get a much richer picture and, as a consequence, a larger set of feasible business models to choose from.

3. Feature-Solution Graphs

3.1. A Quick Introduction

A Feature-Solution (FS) graph captures architectural knowledge in the form of desired features and solutions that realize these features. A typical example of a FS-graph is shown in Figure 1. The features as well as the solutions are captured in AND-OR decompositions. We use an AND decomposition to denote that all constituents of a node are included, an OR to select an arbitrary number of constituents, and an EXOR to select exactly one constituent.

Besides the AND-OR relationships, the FS-graph contains directed *selection* edges (represented by a solid curve that ends with a hollow pointer) to establish the connection between features and solutions. Thus, a feature in the Feature (F) space selects a solution in the Solution (S) space. In some cases, it is useful to outrule a solution explicitly. This is done with *negative selection* (or *rejection*) edges (represented by a dashed curve that ends with a hollow pointer).

The selection and rejection edges establish a rather strong relation between features and solutions, that is, they establish relations of the form *if situation X is encountered, then (don't) apply Y*. In practice, many useful relations that we wish to model are not that clearcut. We will use *positively-influenced-by* (represented by a dash-dotted curve that ends with a harpoon pointer) and *negatively-influenced-by* (represented by a dotted curve that ends with a harpoon pointer) edges to establish weaker relations.

As can be seen in Figure 1, multiple edges may originate from a single node. By default, these edges are AND related, that is, they are all selected simultaneously. However, in some cases, we need to express optional and exclusive relations. This is done by explicitly annotating these relations as such. For instance, the requirement to-be-operational-within-three-months translates to

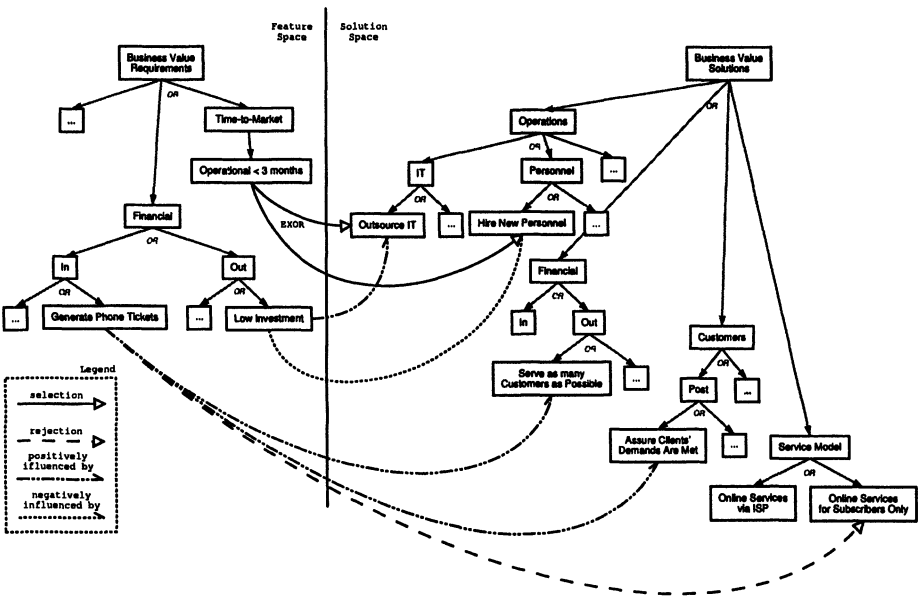

Figure 1. FS-graph example.

either an outsource-IT or a hire-new-personnel solution, as indicated by the EXOR relation.

3.2. Semantics

The semantics can be given informally as follows. First of all, the AND-(EX)OR relations are generalized into a single $[x, y]{-}of{-}N$ relation to select i out of N constituents, with $0 \leq x \leq i \leq y \leq N$. A configuration is defined as a set of activated (i.e., instantiated) nodes as induced by the $[x, y]{-}of{-}N$ relation. The configurations thus obtained are called the degrees of freedom of a FS-graph.

A node in F-space can be connected to a node in S-space by means of the select relation. Such a connection implies that if a source node in a select relation is activated, then the target node is activated as well. As a result, the activated target node selects a particular branch, which may thereby exclude other branches. Thus, the select relation reduces the degrees of freedom. The select relation is transitive. It activates other nodes, which in their turn may activate other nodes. In contrast, the reject relation is *not* transitive since it

does not activate the target node, and therefore the target node cannot activate other nodes. Select relations may lead to contradictions (for instance, two activated nodes nodes in the feature space may select nodes in the solution space that are EXOR-related). This is not a problem, it simply means that a particular configuration is not feasible. Notice that we abstract away from search processes, conflict resolution, order of node activation, and the like. We only define which configurations are feasible and which are not.

The $[x, y]$–of–N and the select/reject are the fundamental relations. The other two relations that we use (i.e., the positively/negatively-influenced-by) merely enrich the model in order to cope with the problem at hand. The semantics of the positively/negatively-influenced-by relations are defined as relations that are only meaningful between *active* nodes in a particular configuration. These are transitive relations: an active node can influence another node (either positively or negatively), which in its turn may influence other active nodes, and so on. We will demonstrate later how the positively/negatively-influenced-by relations are used in conflict resolution.

Another way to give an interpretation of a FS-graph is to view it as a production system consisting of production rules, as used as a knowledge representation formalism in the AI field. Production rules have the form: if ≪a particular situation X is encountered≫ then ≪select solution Y≫. Obviously, select relations in the FS-graph establish such production rules. But also the $[x, y]$–of–N relation can be interpreted as a production rule in the sense that it selects a number of constituents. What sets a FS-graph apart is that it may contain degrees of freedom that can be used to explore design alternatives.

3.3. Advanced Modeling Concepts

Architectural knowledge is often context sensitive, and can only be applied in certain situations. In order to capture contextual information, we augment a FS-graph with an optional Context (C) space. Contextual information in the C-space is organized in AND-(EX)OR decompositions as well.

In Figure 2, we show how the C-space is used to capture stakeholders' views. In particular, the figure shows how conflicting requirements can be incorporated in a FS-graph. From *The Amsterdam Times* point of view, the set-the-price requirement translates into the selection of the originating business model and the rejection of the terminating business model. However, *The Last Mile* stakeholder favors exactly the opposite. In order to model both perspectives in one FS-graph, a conditional node type is introduced, which is visualized as a switch. If a switch node is selected, the switch is closed and establishes context-dependent relations between features and solutions.

An interesting conclusion that can be drawn directly from Figure 2 is that a business model in which *The Amsterdam Times* and *The Last Mile* both select the set-the-price requirement leads to a configuration that is not feasible. As a

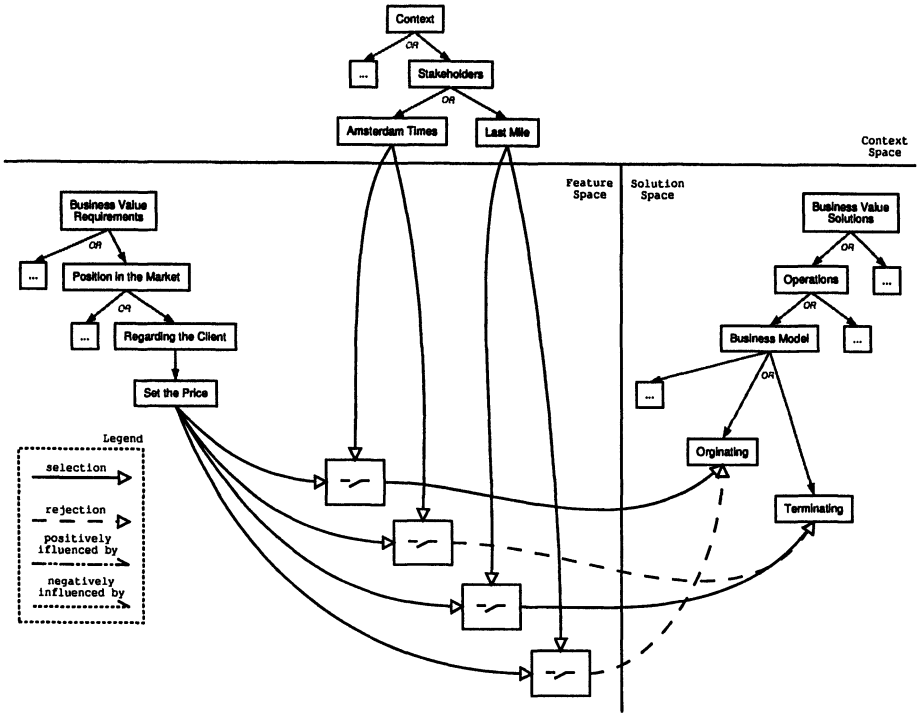

Figure 2. FS-graph example with C-space.

consequence, either stakeholder will have to drop this requirement. In the next section, we explore conflict detection and conflict resolution in more depth.

4. Case Study

In this section, we discuss how a design space can be explored systematically. Particularly, we show how design spaces can be captured and how trade-offs can be identified and dealt with. The approach is illustrated with the e-commerce case introduced in section 2. We have successfully applied the approach in other domains as well, including a multi-channel system for selling railway tickets and WEB servers supporting replicated objects.

4.1. e³-value Framework augmented with FS-graphs

By today's standards, e-commerce projects require a short time to market to obtain or retain a competitive edge. As a consequence, there is no time to go

into a full-fledged system development process. Instead, methods are required to quickly assess whether a newly conceived e-business is feasible or not. The e^3-value framework [11, 12] offers such a lightweight approach. The framework provides three viewpoints that address the concerns of the stakeholders involved in an e-commerce project:

Business value viewpoint. This viewpoint focuses on economic value creation, distribution, and consumption in a multi-actor network.

Business process viewpoint. This viewpoint focuses on how the business value viewpoint is put into operation in terms of business processes.

System architecture viewpoint. This viewpoint focuses on the information system and infrastructure that support the e-business.

As its name suggests, the business value viewpoint is mainly of interest to the top management stakeholders. It enables setting up a prediction of revenues and expenses based on value exchanges between multiple actors. The business process viewpoint focuses on ownership of business processes. It determines which actor is involved in a particular value proposition and in what way. The system architecture viewpoint is usually a concern of an IT department, but it is crucial in the lightweight approach since information systems are a critical success factor in e-commerce and, typically, major investments and operational expenses are involved.

As presented above, the viewpoints are relatively unrelated, which makes it difficult to reason about viewpoint interactions and, hence, to assess the consequences of decisions taken in a particular viewpoint. We therefore add a process step called design space exploration, in which we reason about design alternatives in a systematic way. This step is centered round FS-graphs. For each viewpoint recognized in the e^3-value framework a FS-graph is established. Features (i.e., requirements) are linked to solutions, and the S-space in one viewpoint serves as a F-space of the subordinate viewpoint. In this way, the business value, business process, and system architecture viewpoints are related, which enables us to reason about the impact of decisions made in all viewpoints.

4.2. Exploring the Design Space

An e-business must be profitable to all actors involved. For this reason, we analyze the case for multiple actors. For reasons of space, we restrict ourselves to the main actors only, *The Amsterdam Times* and *The Last Mile*. In addition, our focus will be on the business value viewpoint. After all, this viewpoint is the main driver in setting up a new business. The other two viewpoints are more or less derived from the business value viewpoint.

The exploration of the design space involves the following two steps:

Establish FS-graphs. First, the requirements are captured in the F-space of a FS-graph for each actor involved in the e-business, solutions are codified in the S-space, and the requirements and the solutions are then related from the perspective of each actor.

Establish business cases. Next, business cases are derived from the FS-graph. We use a rather technical definition of a business case. A business case is defined as an economically and technically feasible set of related configurations of a FS-graph as obtained by selecting a particular set of features.

Establish FS-graphs. The business value requirements stem from different sources. Some requirements have been gathered from the commissioners. However, they usually do not have enough experience with e-commerce to articulate their needs accurately [11]. Other requirements have been obtained from past experiences, and yet another set of requirements can be regarded as common sense. The set of requirements that we established is used to model the requirements of *The Amsterdam Times* as well as *The Last Mile*. Some of the requirements are relevant to both actors, whereas others are specific for one actor only. The set of design solutions have been gathered in a similar way. Again, these solutions stem from a mixture of commissioners input, past experiences, and common sense.

The business value requirements and solutions are related by means of a FS-graph. The FS-graph contains all the knowledge that is required for analyzing business cases at the business value viewpoint level. A partially completed FS-graph is shown in Figure 3. Note that the complexity of typical FS-graphs such as the one shown in Figure 3 indicates that we need abstraction mechanisms to show those aspects that currently have our focus of attention. It also indicates that analyzing various business cases without knowledge capturing models such as a FS-graph is infeasible.

Establish Business Cases. Finally, business cases are established and the consequences of implementing a particular business case are assessed. As discussed before, the business cases are addressed mainly at the business value viewpoint level. However, the consequences of choices at this level are assessed at the business process and system architecture level too in order to consider their impact on grounds of economical and technical feasibility.

The analysis of business cases involves the following steps:

1 Develop business cases for each actor involved: (a) make strategic decisions, (b) control feasibility, (c) exclude conflicting requirements, and (d) include harmless requirements.

2 Resolve trade-offs between actors.

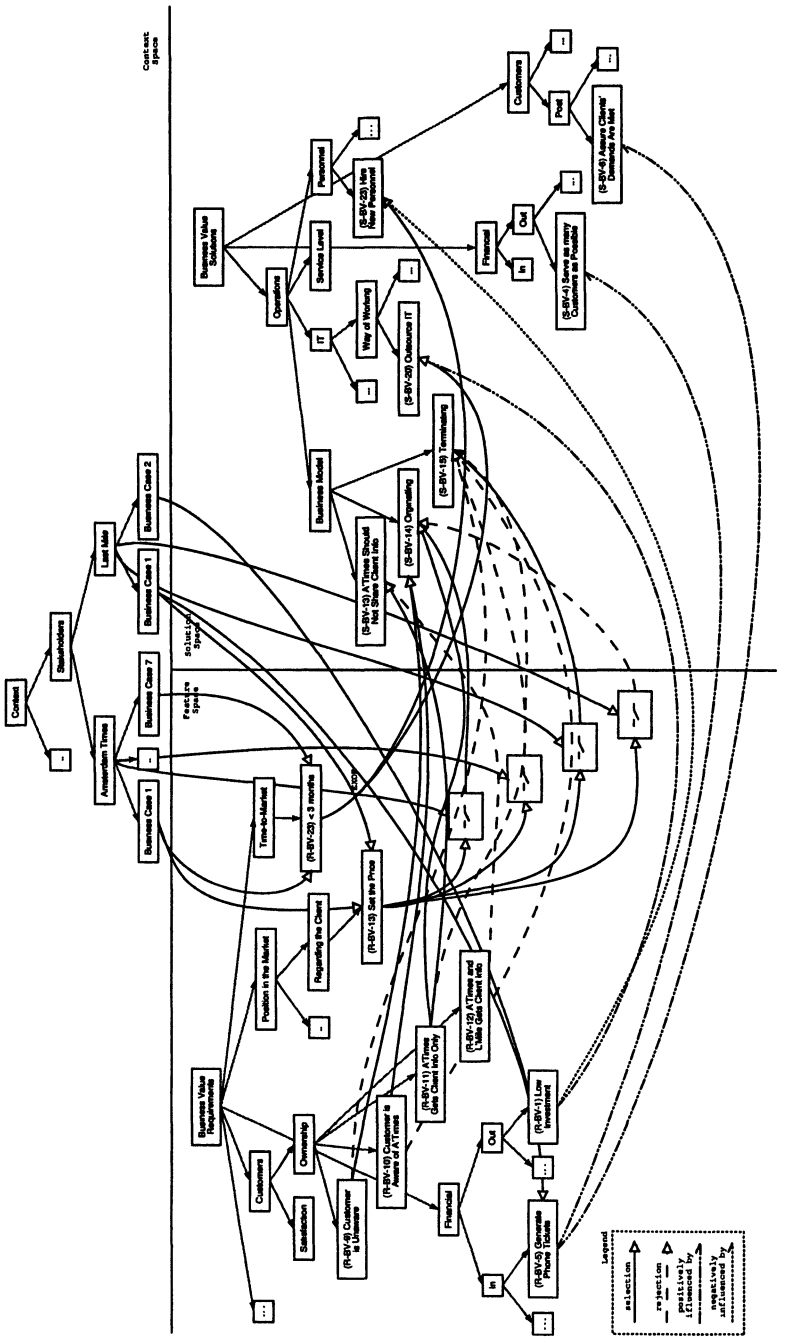

Figure 3. Business value viewpoint FS-graph.

3 Elaborate business cases.

Make Strategic Decisions. The first step in defining business cases is to identify the key strategic issues. A fundamental requirement in e-business activities is a short time to market. This requirement is therefore included in almost all business cases for *The Amsterdam Times*. Other strategic issues that can be identified include: outsourcing services or not, hiring new staff or insisting on not doing so, customer ownership, maintaining little dependence on other parties in the market, and the (exclusive) ownership of setting the price.

The strategic decisions are documented as business-case nodes in the C-space of a FS-graph (see Figure 3). A particular business-case node selects a number of requirements in F-space, which in their turn select or reject particular solutions in S-space. An interesting example is the ownership of setting the price. As discussed before, this is a conflicting requirement. Depending on the stakeholder's viewpoint, the set-the-price requirement translates into the originating or the terminating business model. Therefore, this requirement, when selected simultaneously in business cases for *The Amsterdam Times* and *The Last Mile*, results in infeasible configurations of the FS-graph.

Control Feasibility. The strategic issues are next used to define, for each actor involved, an initial set of business cases each of which exhibits no internal conflicts. This is done on the basis of the knowledge captured in the FS-graph. The following rules are applied for conflict detection and resolution:

- A conflict that involves both a selection and a rejection relation is called a *major conflict*. The existence of such a conflict marks a particular business case as not feasible. The business case is either discarded or re-examined using a slightly different set of strategic decisions as a starting point.
- A conflict which involves an influenced-positively-by and an influenced-negatively-by relation is called a *minor conflict*. The business case is feasible, but requires compromises.
- A conflict which involves either a selection or a rejection, but not both, is classified as either a major or a minor conflict, depending on a further analysis of its impact.

Exclude Conflicting Requirements. For each business case, requirements that are not of strategic importance but whose selection implies a conflict are explicitly excluded from the business case.

Include Harmless Requirements. By the same token, requirements can be added to a business case if they cause no conflict.

Resolve Trade-Offs between Actors. The combination of strategic issues and the application of feasibility control results in seven feasible business cases for *The Amsterdam Times* and two feasible ones for *The Last Mile*. Next, the business cases for each actor involved are compared. Basically, the same strategy for conflict resolution is followed. We determine whether a particular combination has no conflicts or has minor conflicts only. Such a combination is a feasible business case for both actors involved, and can be elaborated further, as discussed in the next paragraph.

By systematically analyzing the 14 business case combinations, we found three promising combinations. However, in all three business cases, *The Amsterdam Times* has to give up customer ownership, which is a major compromise from the viewpoint of *The Amsterdam Times*. Once customer ownership is given up by *The Amsterdam Times*, we have the choice of adopting the originating as well as the terminating business model. The actual choice depends on whether particular strategic issues are emphasized or not as reflected in the business cases identified for the *The Amsterdam Times* and *The Last Mile*.

Elaborate Business Cases. After having identified the most promising business cases, the subsequent steps towards realizing the e-business are straightforward. First, business process and system architecture view models are established. Again, the FS-graph is the central concept in this step, since at each viewpoint level requirements are linked to possible solutions. In [8, 9], we describe how architectures can be generated from a FS-graph. These generated architectures are then assessed with respect to given quality requirements, such as performance and security, but also profitability (see for instance [10]). Finally, the most appropriate system architecture is realized, and we are in business.

4.3. Lessons Learned

The e^3-value framework provides the means to assess an e-business idea from three viewpoints corresponding with the interests of the various stakeholders involved. As presented in [10], the three viewpoints are not connected strongly, which makes it difficult to trace design decisions from business concepts all the way down to system architectures and eventually the e-business system itself. FS-graphs fill in this void. They are used to relate features and solutions within and between viewpoints. As a result, design decisions are now traceable, e.g., by coloring the links in FS-graphs that make up the business case chosen. But even more importantly, the FS-graph approach allows us to reason about solutions systematically. Conflicting stakeholder views are detected and design alternatives are assessed to make sure that a particular solution is acceptable to all parties involved.

The exercise did not result in unexpected business models, or provide the expert that devised the models as presented in [10] with new architectural insights. The approach, though, did result in a much more thorough and systematic exploration of the design space. In particular, the analysis of the business cases yielded two interesting observations that were not clear from the outset:

1 *The Amsterdam Times* is dependent on other parties in order to meet the short-time-to-market requirement that is present in almost all of its business cases.

2 *The Amsterdam Times* can only assure a high level of customer ownership in combination with the originating business model.

Thus, the impact of business cases stands out more clearly by applying a systematic analysis than would otherwise be possible by an ad-hoc approach.

5. Related Work

A Design Rationale (DR) [4, 19] is a representation of the reasoning behind the design of an artifact. It is concerned with methods and representations for capturing why designers have made the decisions they have made. A well-known approach to representing design rationale is Design Space Analysis, whose notation is called QOC (Questions, Options and Criteria) notation [18]. Questions in QOC are key design issues, and Options are possible answers to the Questions. The solution fragments of the FS-graph contains both Questions and Options. Criteria in QOC are used to choose between Options. They resemble the requirements as captured in the FS-graph. Options and Criteria in QOC are linked by Arguments such as *supports* or *objects to* which resemble the links between the requirements and solution part of the FS-graph. DR is most often used as a tool in the design process, especially the user-interface design process, to augment design reasoning, and to help in formulating and communicating arguments. It usually pertains to one particular set of choices, not to a complete space of design options, as we try to capture in the FS-graph. In [3], design spaces are used to map requirements to components. The approach is geared towards the reuse of components within a domain, and does not handle conflicts.

In Feature-Oriented Domain Analysis (FODA) [14, 6], and variants thereof, a family or product line is represented in a feature tree. Features can be mandatory, alternative, or optional. A specific product is then composed by choosing a set of alternative and optional features; these express the variabilities within the product line. The feature tree may span the current set of products, or a design space of possible products, or a mixture. The feature tree thus resembles the F-part of the FS-graph. Usually though, features of a product line are units of functionality rather than quality concerns.

In requirements engineering, viewpoints are used to express different perspectives of stakeholders, agents that may have different responsibilities or roles [7]. Some requirements engineering methods, and the associated tool environments, treat viewpoints as first class objects that may be used and manipulated by the users of that method. At an early stage, each viewpoint is represented. Later on, viewpoints are compared and conflicts and inconsistencies are generally solved to arrive at a unified, integrated viewpoint. Often, only the chosen alternative is well-documented. Important issues in this area include the handling of inconsistent and conflicting viewpoints [1, 2], and the prioritization of requirements and stakeholder win-conditions [22]. The emphasis is on identifying and resolving conflicts, rather than retaining a picture of the complete space of options.

Early approaches to requirements engineering focussed on eliciting and documenting the requirements sec, and not the reasons for them. In goal-oriented requirements engineering [17, 20], the relation between goals and requirements is represented explicitly. Recently, representation schemes used in goal-oriented requirements engineering have also been used to represent dependencies between quality goals and architectural styles [5].

The Architecture Tradeoff Analysis Method (ATAM) [15] supports the evaluation of software architectures with respect to different quality attributes. In later publications ([16]), the focus of ATAM shifted to architectural approaches and their properties, captured in Attribute-Based Architectural Styles (ABASs). The key of such an analysis is to 'understand the consequences of architectural decisions with respect to the quality requirements of the system'.

The gap between requirements engineering and architecture is also addressed in the CBSP approach [13]. Software requirements may contain information that is relevant to the software architecture, and CBSP supports the task of identifying and elaborating this information. CBSP uses an iterative process of selecting essential requirements, relating these requirements to architectural artefacts, and the identification and resolution of conflicts, until a satisfactory model is obtained. A similar weaving of requirements engineering and architecture development is described in the twin peaks model in [21].

6. Concluding Remarks

We have discussed FS-graphs as a means to document and analyze architectural trade-offs that arise from conflicting stakeholders' concerns. In our experience, a FS-graph is a powerful representation scheme for capturing architectural knowledge. First of all, a sharp distinction is made between Feature (F) and Solution (S) spaces. In this way, the relations between requirements and solutions can be made explicit. This idea can be applied recursively. For instance, in the e^3-value framework, three viewpoints are recognized, which can all be modeled with FS-graphs. The S-space of one viewpoint serves as a

F-space for a subordinate viewpoint. This provides the means to trace requirements at the business level all the way down to the system architecture and its realization. Secondly, a FS-graph is typically underspecified in the sense that not all nodes in F-space have a connection with nodes in S-space. To put it differently, a FS-graph has degrees of freedom that can be used to explore design alternatives. Thirdly, context-sensitive knowledge, such as stakeholders' viewpoints, can be modeled in a Context (C) space. We introduced a conditional node type (i.e., a switch) for establishing context-dependent relations between features and solutions.

We have recently started the QUASAR project, which stands for QUAlity-driven Software ARchitecture. The goal is to generate software architectures and implementations from domain and application specific knowledge captured in the FS-graph. In the case of e-commerce projects, this would mean that a feasible e-business solution can be elaborated directly into an operational system. Besides generation techniques, the QUASAR project concentrates on visualization techniques and abstraction mechanisms to focus on specific parts of a FS-graph. Additionally, we look at automatic reasoning with FS-graphs. In particular, we want to support conflict (trade-off) detection and conflict resolution using the principles discussed in this paper.

Future work includes the exploration of the concept of context in FS-graphs in more depth. In this paper, we used the Context (C) space to model, possibly conflicting, stakeholder views. An interesting option is to enrich the C-space with a notion of configuration management. We envisage that the C-space can be used to document particular configurations of features and solutions that realize these features. In this way, we may capture variations in a product family. Another possibility is to define evolutionary migration trajectories. Each step in a trajectory can be captured in a configuration. A FS-graph can then be used to guide us through successive steps in a migration process.

References

[1] Special Section: Managing Inconsistency in Software Development. *IEEE Transactions on Software Engineering*, 24(11):906–1001, 1998.

[2] Special Section: Managing Inconsistency in Software Development. *IEEE Transactions on Software Engineering*, 25(6):782–870, 1999.

[3] L. Baum, M. Becker, L. Geyer, and G. Molter. Mapping Requirements to Reusable Components using Design Spaces. In *Proceedings 4th International Conference on Requirements Engineering*, pages 159–167. IEEE, 2000.

[4] Simon Buckingham Shum and Nick Hammond. Argumentation-based design rationale: What use at what cost? *International Journal of Man-Machine Studies*, 40(4):603–652, 1994.

[5] L. Chung, D. Gross, and E. Yu. Architectural design to meet stakeholder requirements. In P. Donohue, editor, *Software Architecture*, pages 545–564. Kluwer Academic Publishers, 1999.

[6] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, Massachusetts, 2000.

[7] Peta Darke and Graeme Shanks. Stakeholder Viewpoints in Requirements Definition: A Framework for Understanding Viewpoint Development Approaches. *Requirements Engineering Journal*, 1:88–105, 1996.

[8] Hans de Bruin and Hans van Vliet. Scenario-based generation and evaluation of software architectures. In Jan Bosch, editor, *Proceedings of the Third Symposium on Generative and Component-Based Software Engineering (GCSE'2001), Erfurt, Germany*, volume 2186 of *Lecture Notes in Computer Science (LNCS)*, pages 128–139, Berlin, Germany, September 10–13, 2001. Springer-Verlag.

[9] Hans de Bruin and Hans van Vliet. Top-down composition of software architectures. In Per Runeson, editor, *Proceedings of 9th International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'2002), Lund, Sweden*, pages 1–10, April 8–11, 2002.

[10] Jaap Gordijn and Hans Akkermans. e^3-value: Design and evaluation of e-business models. *IEEE Intelligent Systems*, 16(4):11–50, 2001. Special issue on e-business.

[11] Jaap Gordijn, Hans Akkermans, and Hans van Vliet. Value based requirements creation for electronic commerce applications. In *Proceedings of HICSS 33*, Hawaii, USA, 2000.

[12] Jaap Gordijn, Hans de Bruin, and Hans Akkermans. Scenario methods for viewpoint integration in e-business requirements engineering. In *Proceedings of HICSS 34*, Hawaii, USA, January 3–6, 2001.

[13] Paul Grünbacher, Alexander Egyed, and Nenad Medvidvic. Reconciling Software Requirements and Architectures: The CBSP Approach. In *Proceedings 5th IEEE International Symposium on Requirements Engineering*, pages 202–211. IEEE, 2001.

[14] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, 1990.

[15] R. Kazman, M. Barbacci, M. Klein, and S.J. Carriere. Experience with Performing Architecture Tradeoff Analysis. pages 54–63, 1999.

[16] R. Kazman, M. Klein, and P. Clements. ATAM: Method for Architecture Evaluation. Technical Report Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Pittsburg, USA, 2000.

[17] Axel van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Conference Proceedings ICSE'00*, pages 5–19, Limerick, Ireland, 2000. ACM.

[18] Allen MacLean, Richard M. Young, Victoria M.E. Bellotti, and Thomas P. Moran. Questions, options and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6(3 & 4):201–250, 1991.

[19] T.P. Moran and J.M. Carroll, editors. *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1994.

[20] John Mylopoulos, Lawrence Chung, Stephen Liao, Huaiqing Wang, and Eric Yu. Exploring alternatives during requirements analysis. *IEEE Software*, 18(1):92–96, January 2001.

[21] Bashar Nuseibeh. Weaving Together Requirements and Architectures. *IEEE Computer*, 34(3):115–117, 2001.

[22] Jung-Won Park, Daniel Port, and Barry Boehm. Supporting Distributed Collaborative Priorization for WinWin Requirements Capture and Negotiation. In *Proceedings 3rd World Multi Conference on Systemics, Cybernetics and Informatics (SCI'99)*, pages 578–584. IIIS, 1999.