Modelling Technical Constraints and Preconditions for Alternative Design Decisions *

Alexander Pacholik, Matthias Riebisch
Ilmenau University Technology
{matthias.riebisch | alexander.pacholik}@tu-ilmenau.de

Abstract: For architectural design decisions, a high number of goals and constraints has to be considered. For a proper management of their complexity and as a precondition for tool support, they have to be modeled explicitly. Existing approaches fail to provide a predictable, rigor decision-making as well as a to deal with incomplete or even partly contradicting information. This paper presents an approach for an explicit modeling of constraints for architectural reasoning and of preconditions for existing solutions based on the concept of technical terms. The preconditions are modeled within the Goal Solution Scheme, and are combined with information about the impact of existing solutions on quality goals.

1 Introduction

Development and evolution of software systems are highly complex tasks that have to be performed with a high efficiency. Particularly, design decisions are complex because a high number of requirements and constraints have to be considered [HKN+07], with some of them being conflicting, vague or even undefined. Wrong decisions lead to high effort for revision. Furthermore, the utilization of solutions instruments – such as architectural patters, styles, frameworks, and evolution patterns in form of refactorings – is important for the efficiency of the development process, and for the quality of the resulting design. Many of them demand for preconditions for applicability, for example the provision of a platform, or the availability of a service. These preconditions have to be considered in design decisions as well. Most of them are of a technical nature. Embedded systems have to meet even more kinds of constraints than other systems, for example regarding platform, energy consumption, and cost [Kop08].

Different kinds of goals, requirements, and constraints have been considered by design methodologies. While functional ones are obviously in the focus of customers, newer architectural design approaches concentrate on quality goals when selecting existing solution instruments, for example the Goal Solution Scheme (GSS) [BR10] explained in section 3. However, the consideration of constraints and preconditions is not yet sufficiently solved for complex design situations, as they are typical for embedded system design.

*This research has been partly funded by the federal state Thuringia and the European Regional Development Fund ERDF through the Thüringer Aufbaubank under grant 2007 FE 9041 and under grant 2010 FE 9094.

The contribution of this paper consists in an explicit representation and evaluation of constraints and preconditions for the selection of existing solution instruments in order to support decision-making and problem-solving during design. The new concept extends the goal-oriented decision-making approach of the GSS. The formalization is achieved by using technical terms as references, and a set of operators to evaluate the satisfaction of constraints and preconditions. The evaluation can be performed by tools, thus reducing the complexity and error-proneness of decision-making.

2 Related Work

As a source of solution instruments, *patterns and styles* [HA07] are used. The architect's set of solution instruments is frequently collected in form of a toolbox, thus representing a knowledge base of design knowledge. A toolbox should contain a catalogue of approved methods and solution templates (e.g. patterns and tactics [BKB02]), as well as a catalogue of fundamental technologies and tools (e.g. frameworks).

For a *goal-oriented way of selecting* solution elements, the impact of the toolbox elements on quality properties is required as a decision criterion. Usually, pattern catalogues (e.g. [BMR+96, GHJV94]) provide descriptions for context, problem, and solution. The context part covers some of the needed information on preconditions. However, an impact on quality properties of the resulting architecture is not provided as a decision criterion. Influences on quality properties are considered as impact relations by our prior works [BR10].

Design constraints restricting the design or the solution space are described by [TvV09]. However, the relation between constraints and requirements as well as the modelling of preconditions for solution instruments requires for further elaboration. Modelling restrictions can be expressed by the Object Constraint Language (OCL) [LO03]. For the definition of preconditions and constraints, ontologies are applied as means of description [BL10]. The need for a firm set of relationships, however, contradicts to the open character of an ontology but is required for evaluation. The concept of technical terms [TNAKN11] constitutes another candidate for defining preconditions and constraints.

3 Design Decision Procedure with the Goal Solution Scheme

For design decisions regarding reusable solution instruments, goals and constraints have to be fulfilled, and applicable solution instruments have to be identified. A set of solutions instruments is usually collected by architects and is therefore referred to as architect's toolbox. Such a toolbox may cover architectural patters, styles, frameworks, evolution patterns in form of refactorings, and even heuristics and design principles. Design decisions regarding the selection of solution instruments demand for a mapping of the goals and constraints – which are part of the problem space – onto solution instruments – which are part of the solution space – along with the impact of solution instruments on quality goals and preconditions for applicability. Recently, the Goal Solution Scheme (GSS) [BR10] was developed to represent this mapping between problem and solution space regarding

quality goals. In the GSS, this mapping is represented explicitly by impact relationships between solution instruments and quality goals. Different layers have been introduced (see Figure 1) to reduce the gap between goals and solution instruments by goal refinement. The GSS achieves a further reduction of the gap by introducing design principles (layer III) to represent the impact relationships independently of concrete circumstances of a project.

An evaluation of the applicability of solution instruments stored in the toolbox can be performed, if the constraints for a specific design decision are compared with the preconditions of the solution instruments. This evaluation is the first part in the decision making chain, which is typically embedded into the incremental design process according to the specific conditions of a software project.

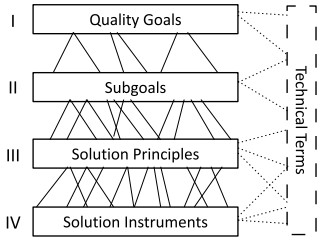

Figure 1: Layers of the Extended Goal Solution Scheme

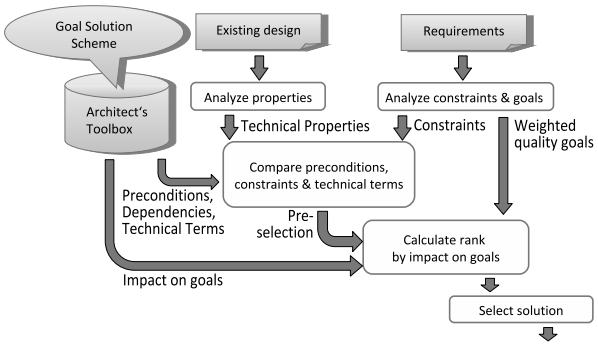

Figure 2: Decision Procedure with Two Selection Steps

Figure 2 depicts the general procedure of selecting solution instruments as candidate solutions within one design cycle. The preselection is performed as the first step, which which enables the reduction of the set of solution instruments to the applicable ones. Its input is acquired by analysing constraints stemming from the existing design and constraints defined by requirements. Missing information is collected interactively from the designer. To perform the preselection, constraints are compared to the preconditions of solution instruments. In the second step, the preselected solution instruments are ranked according to their impact on the desired quality goals, in order to propose the most suitable solutions to the architect. This two-step selection procedure is supported by the open source tool QUARC as a part of the EMFTrace suite [EMF].

4 Constraints, Preconditions and Technical Properties

For the second step, impact relationships are sufficiently covered by the concepts of the GSS. However, *constraints and preconditions* and thus, restrictions and interdependencies between solution instruments demand for an explicit representation to enable the preselection as the first step. This representation is required to manage the complexity of decision-making, and to provide a foundation for method and tool support of this preselection step.

In this paper we focus on *design constraints* regarding technology and resource aspects, and their influence on architectural design decisions as restrictions of the solution space. A constraint constitutes a restriction which must be satisfied by a viable design solution.

Preconditions express requirements for the applicability of a solution instrument in terms of e.g. expected services, platforms, resources or standards. Furthermore, preconditions can be used to express interdependencies between the application of solution instruments, e.g. caused by a conflict in resource consumption. The evaluation of preconditions and constraints in the preselection step described above requires an assessment of the compliance with information on the current and future design that we refer to as *technical properties*. The concept of *technical properties* provides a common foundation for both preconditions and constraints and expresses are used to express the properties *provided* by a solution instrument, e.g. provided interfaces, services, features definitions, and resources that can satisfy the preconditions of other solution instruments. Each technical property is bound to a *technical term*. Compared to the GSS, the *technical terms* constitute a semantic layer, which contains facts used for evaluating restrictions and interdependencies within the solution space.

Technical terms are semantically defined in the solution space or in the application domain, in order to enable reasoning about the design. A technical term is composed of a unique identifier and type information. The actual value is contained an associated technical property object, as depicted in in Figure 3. The technical property is an OCL LiteralExpression as will be explained in Section 5, and contains the desired value for the technical term. A precondition refers to one or multiple technical term and contains instructions for their further evaluation. Not until evaluation, the associated technical properties are accessed.

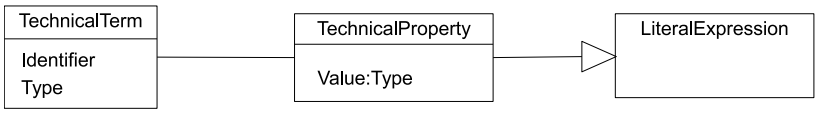

Figure 3: Implementation of Technical Terms and Technical Properties

5 Formalization of Constraints and Preconditions

For the definition of constraints and preconditions, the Object Constraint Language (OCL) is used, because the OCL is widely accepted as an industrial standard, and it is supported by various tools. The OCL is by concept a declarative, side effect free language.

Technical properties are defined as OCL literal expressions, which when evaluated, return a primary data type, such as Boolean, Integer, Real, String or a Collection data type. For the definition of constraints and preconditions we use the set of OCL's pre-defined operations for these standard types, such as boolean operators, arithmetic operators, comparators, and collection operators. The operators can be used to evaluate the technical properties and to build complex OCL expressions for constraints and preconditions, whose evaluation results in a Boolean type. The mapping from *technical terms* of the GSS to OCL

expressions is depicted in Figure 3. The *technical property* itself is an OCL literal expression, which can be accessed by an OCL constraint expression using the *technical term* as a reference object, which already contains the type information.

Figure 4 shows example preconditions between solution instruments on the left side and technical terms, written in curled braces, on the right side. Constraints are structurally visualized by a dotted arrow with the constraint annotated in curled braces. The OCL text is omitted for pure *required* dependencies. Figure 4 contains four example preconditions for solution instruments with references to technical terms of different types.

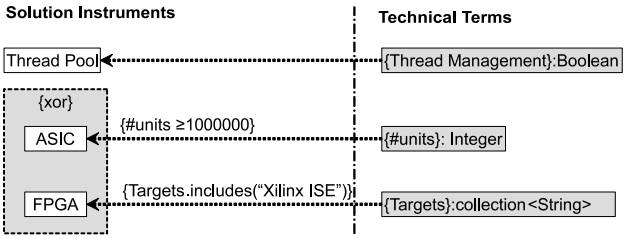

Figure 4: Examples for Technical Terms and Constraints

The first precondition denotes that the *Thread Pool* solution instrument can only be selected, if a *Thread Management* service is available. This precondition refers to the technical term *Thread Management*, representing a service requirement. It is evaluated using a boolean operator: If there is a property provider for the *Thread Management* capability – either in the current design or in a candidate for solution instrument – the precondition evaluates to true. The second precondition expresses that the *ASIC* solution instrument requires a production volume of at least 1 million number of units. The technical term *#units* refers to a technical property of type integer. In general, such numeric properties are well suited to describe resource requirements, e.g. for computing or memory. The third precondition example for the *FPGA* solution instrument states, that the *"Xilinx ISE"* development environment constitutes a supported target for model-based code generation for FPGAs. It references the technical term *Targets*. To enable sets of requirements, a technical property with the type `collection<strings>` is applied. The OCL collection operator `includes` is used to express the condition. The evaluation of the technical property returns a collection of supported targets. OCL Collection types and operators can be used to model provided capabilities and characteristics. A fourth precondition expresses a mutually exclusive selection between the solution instruments *ASIC* and *FPGA*, visualized by the shaded box, containing the $\{xor\}$ keyword.

6 Conclusion and Future Work

The presented approach provides an explicit modelling of complex preconditions and constraints that have to be considered during design decision on architectural level. These preconditions and constraints are expressed by references to so-called *technical properties*, which are evaluated by expressions. The technical properties refer to the concept of

technical terms, which have been introduced to provide the semantic definition. Technical properties, technical terms and expressions are formally defined using the OCL.

Future works based on the approach include the extension to implementation constraints, to cover decisions regarding code evolution and reengineering. For a semantic definition of the technical terms, ontologies are a potential concept that has to be investigated. Using this concept, references between terms can be expressed by ontology relations.

References

[BKB02] L. J. Bass, M. Klein, and F. Bachmann. Quality Attribute Design Primitives and the Attribute Driven Design Method. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 169–186. Springer, 2002.

[BL10] M. Bennicke and C. Lewerentz. Towards Managing Software Architectures with Ontologies. In G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, editors, *Graph transformations and model-driven engineering*, pages 274–308. Springer, 2010.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1 edition, July 1996.

[BR10] S. Bode and M. Riebisch. Impact Evaluation for Quality-Oriented Architectural Decisions regarding Evolvability. In M. Babar and I. Gorton, editors, *Proc. 4th European Conference on Software Architecture, ECSA 2010*, pages 182–197. Springer, 2010.

[EMF] EMFTrace repository suite. https://pix.theoinf.tu-ilmenau.de/trac/EMFTrace/.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Softwaresystemen*. Addison-Wesley Professional, 1994.

[HA07] N. Harrison and P. Avgeriou. Pattern-Driven Architectural Partitioning: Balancing Functional and Non-functional Requirements. In *Second International Conference on Digital Telecommunications, 2007 (ICDT '07)*, pages 21–26. IEEE, July 2007.

[HKN+07] C. Hofmeister, P.e Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, January 2007.

[Kop08] H. Kopetz. The Complexity Challenge in Embedded System Design. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 3 –12, may 2008.

[LO03] S. Loecher and S. Ocke. A Metamodel-Based OCL-Compiler for UML and MOF. In *OCL 2.0 - Industry standard or scientific playground?, Proc. 6th International Conference on the UML and its Applications, UML 2003 in ENTCS*, October 2003.

[TNAKN11] A. Tamrawi, T. Th. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen. Fuzzy set and cache-based approach for bug triaging. In T. Gyimóthy and A. Zeller, editors, *SIGSOFT FSE*, pages 365–375. ACM, 2011.

[TvV09] A. Tang and H. van Vliet. Modeling constraints improves software architecture design reasoning. In *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009.*, pages 253 –256, September 2009.