Methodological Support for Requirements Elicitation and Formal Specification

Maritta Heisel
Fakultät für Informatik
Universität Magdeburg
D-39016 Magdeburg, Germany
heisel@cs.uni-magdeburg.de

Jeanine Souquières
LORIA—Université Nancy2
B.P. 239 Bâtiment LORIA
F-54506 Vandœuvre-les-Nancy, France
souquier@loria.fr

We propose a method for the elicitation and the expression of requirements. The requirements can then be transformed in a systematic way into a formal specification that is a suitable basis for design and implementation of a software system. The approach – which distinguishes between requirements and specifications – gives methodological support for requirements elicitation and specification development. It does not introduce a new language but builds on known techniques.

1 Introduction

The usefulness of formal specification is more and more accepted by researchers and practical software engineers. But formal specification techniques still suffer from two drawbacks. First, research spends more effort to develop new languages than to provide methodological guidance for using existing ones. Often, users of formal techniques are left alone with a formalism for which no explicit methodology has been developed.

Second, formal specification techniques are not well integrated with the analysis phase of software engineering. Often, formal specification begins with a very short description of the system to be implemented, and detail is added during the development of the formal specification. Such a procedure does not adequately take into account the need to thoroughly analyze the system to be implemented and the environment in which it will operate *before* a detailed specification is developed. The elicitation of the requirements for the system and their transformation into a formal specification are separate tasks.

Our approach treats both of these issues. It introduces an explicit requirements elicitation phase, the result of which is an adequate starting point for the development of the formal specification. Furthermore, it provides detailed methodological guidance for requirements elicitation as well as for specification acquisition. The different phases provide feedback to one another: not only is the specification based on the requirements, but the specification phase may also reveal omissions and errors in the requirements.

We propose a formal expression of requirements. After a few informal brainstorming steps, the requirements are formalized as constraints on sequences of events or operations that can happen or be invoked in the context of the system. Thus, the transition from informal to formal means of expression is performed very early in the software development process. This has the advantage that requirements can be analyzed, e.g., for consistency, and that a formal notion of correctness of a specification with respect to given requirements becomes possible.

2 Requirements Elicitation

Our approach to requirements engineering is inspired by object oriented methods such as Fusion [CAB+94] or OMT [RBP+91], and by the work of Jackson and Zave [JZ95, ZJ97].

The starting point for requirements elicitation is a brainstorming process where the application domain and the requirements are described in natural language. This informal description is then transformed into a formal representation.

Our approach is suitable for transformational as well as reactive systems. Transformational systems offer a set of system operations that can be invoked by the user. Reactive systems, on the other hand, have to react to events that happen in the environment.

The difference between requirements and a specification is that requirements refer to the entire system to be realized, whereas a specification refers only to the part of the system to be implemented by software. To express requirements formally, we use *traces* of the system, i.e., sequences of events that happen in certain states and at a certain time. For transformational systems, events can be identified, too, namely the invocation and the termination of the system operations. In this way, also systems that are partially reactive and partially transformational, can be treated.

Requirements elicitation is performed in six steps, which provide methodological guidance for analysts. The

concept of an *agenda* [Hei98] is used to express our method. An agenda is a list of steps to be performed when carrying out some task in the context of software engineering. The result of the task will be a document expressed in a certain language. Agendas contain informal descriptions of the steps. With each step, schematic expressions of the language in which the result of the activity is expressed can be associated. The schematic expressions are instantiated when the step is performed. The steps listed in an agenda may depend on each other. Usually, they will have to be repeated to achieve the goal, because later steps will reveal errors and omissions in earlier steps.

Agendas are not only a means to guide software development activities. They also support quality assurance because the steps of an agenda may have validation conditions associated with them. These validation conditions state necessary semantic conditions that the artifact must fulfill in order to serve its purpose properly.

In the following, we list the steps of the agendas we have developed for requirements elicitation and transforming requirements into formal specifications. Only the most important validation conditions are mentioned.

The steps of the agenda for requirements elicitation are:

1. Introduce the domain theory.
   All necessary notions must be introduced. These can either be entities, corresponding to nouns in a natural-language description, or relationships, corresponding to verbs in a natural-language description.
2. List all possible events that can happen in connection with the system, together with their parameters.
3. Classify the events as: (i) controlled by the environment and not shared with the software system, (ii) controlled by the environment but observable by the software system, (iii) controlled by the software system and observable by the environment, and (iv) controlled by the software system and not shared with the environment.
   Validation condition: There must not be any events controlled by the software system and not shared with the environment.
4. List possible system operations that can be invoked by users, together with their input and output parameters. Introduce a relation between the input and output parameters.
5. State the facts, assumptions, and requirements concerning the system in natural language.
   It does not suffice to just state requirements for the system. Often, facts and assumptions must be introduced to make the requirements satisfiable. *Facts* express things that always hold in the application domain, regardless of the implementation of the software system. Other requirements cannot be enforced because e.g., human users might violate regulations. These conditions are expressed as *assumptions*.
6. Formalize the facts, assumptions, and requirements as constraints on the possible traces of system events.

Using constraints to talk about the behavior of the system has the following advantages:

- It is possible to express *negative* requirements, i.e., to require that certain things do not happen.
- It is possible to give scenarios, i.e., example behaviors of the system.
- Giving constraints do not fix the system behavior entirely. They do not restrict the specification unnecessarily. Any specification that fulfills the constraints is admitted.

An example illustrating our approach is given in [HS98].

3 Specification development

Jackson and Zave [JZ95] consider specifications as special kind of requirements. A requirement is a specification if all actions constrained by the requirement are controlled by the software system, and all information it relies on is shared with the software system and refers only to the past, not the future. Requirements (and thus specifications) do not talk about the state of the software system. In contrast to this view, we consider a specification to be a model of the software system to be built in order to satisfy the requirements. It is developed gradually by expressing properties of the system and adding more and more details. Our goal is to support specifiers in building this model, starting from requirements acquired as described in the previous section. A specification may - in contrast to the requirements - make statements about the software system, because it serves as the basis for further design and implementation.

While requirements elicitation is independent of the specification language that is used, the development of a specification depends to a certain extent on the specification language and its means of expression [HS98]. In the following, we assume that the specification language Z [Spi92] is used.

The starting point of the specification development is the whole material obtained by the requirements elicitation phase presented in Section 2. Again, our method for specification acquisition is expressed as an agenda, consisting of the steps

1. Define a first approximation of the state of the software system and the initial states.
2. Augment the specification, incorporating the requirements one by one.

Validation condition: The constraints expressing facts must not be violated.

Step 1 his step consists in giving a first approximation of the system state in such a way that as many as possible of the predicates and functions on the system state that were introduced in the requirements elicitation process can be defined.

For Step 2, the basic idea is to define a Z operation for each event identified in Step 2 of the requirements elicitation phase that is shared with the software system (according to the classification made in Step 3) and for each system operation identified in Step 4. This step should be performed following the sub-agenda:

1. List the events occurring in the constraint.
2. For each event in the list, make a first definition of the corresponding Z operation, or adjust an already existing operation.

Step 1 can be performed by a simple syntactic inspection of the constraint in question. Step 2, however, can be complex with several revisions of the current version of the specification [LS97]. The state of the system and the operations have to be re-considered in order to take into account the evolution introduced by new constraints. Propagation of modifications are important. Incorporating a new constraint may involve the following modifications: (i) adding or modifying state components, (ii) adding or modifying data types, (iii) adding or modifying the state invariant, and (iv) propagating those modifications into the current state of the specification.

The agendas for requirements elicitation and specification acquisition provide an integrated approach that introduces formality as early as possible in the software engineering process. The formal expression of requirements and facts as constraints guide the development of the formal specification. Our approach even allows to define a notion of correctness of a specification with respect to requirements, facts, and assumptions. First, one defines the set of possible traces of the specification as the set of traces where each operation is executed only if its precondition is satisfied. Assuming that the assumption constraints are satisfied, we must show that the set of possible traces induced by the specification fulfills the constraints stated as requirements and facts.

4 Conclusion

The distinguishing features of our approach are the following:

- We introduce a clear distinction between requirements/requirements elicitation on the one hand and specifications/specification acquisition on the other hand.
- We give detailed methodological guidance for requirements elicitation and specification acquisition. The two activities are integrated smoothly.
- We propose a standardized way of expressing facts, assumptions, and requirements. Constraints on the set of possible traces are a very flexible and powerful means of describing a system and its interaction with the environment.
- We do not invent a new language or a new formalism, but instead build on and combine existing approaches.
- The process of requirements elicitation is independent of the specification language to be used.
- We can treat transformational as well as reactive systems. Behavior and data transformation are the most important aspects of computerized systems. Also real-time considerations are taken into account.
- Expressing requirements as constraints on traces makes it possible to define a formal notion of correctness of a specification with respect to the requirements.

References

[CAB+94] D. Coleman, P. Arnold, St. Bodoff, Ch. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The Fusion Method*. Prentice Hall, 1994.

[Hei98] Maritta Heisel. Agendas – a concept to guide software development activites. In J. Bishop and N. Horspool, editors, *Proc. Systems Implementation 2000*, 1998. to appear.

[HS98] M. Heisel and J. Souquières. A Method to Express Requirements and Transform them into a Formal Specification. Technical report, Loria, Nancy (Fr), 1998.

[JZ95] M. Jackson and P. Zave. Deriving Specifications from requirements : an Example. In ACM, editor, *Proc. ICSE'95*, 1995.

[LS97] N. Lévy and J. Souquières. Modelling Specification Construction by Successive Approximations. In *6th Int. AMAST conf.*, Sydney (A), 1997. Springer Verlag.

[RBP+91] J. Rambaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall International. Englewood Cliffs, New Jersey, 1991.

[Spi92] J. M. Spivey. *The Z Notation – A Reference Manual*. Prentice Hall, 2nd edition, 1992.

[ZJ97] P. Zave and M. Jackson. Four dark corners for requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, January 1997.