Architecture: Requirements + Decomposition + Refinement

Maria Spichkova
spichkov@in.tum.de

Abstract

This paper focuses on the system requirements and architecture w.r.t. their decomposition and refinement: how the refinement-based verification can be used to optimize verification process, and which influences it has on the specification process. We introduce here specification decomposition methods, applying which ones can not only to keep the specification readable and manageable, but also find out a number of inconsistencies and underspecifications during specification phase as well, without starting a formal verification process.

1 Motivation

The correctness of a system according to a given specification is essential, especially for safety-critical applications. A formal specification is in general more precise than a natural language one, but a formal specification can as well contain mistakes or disagree with requirements: it is not enough to have detached formal specifications, we also need to validate and to verify them to be sure that the specification conforms to its requirements.

In this paper we focus on the formal specification phase: on requirements specification and on the developing of a logical system architecture and on the corresponding system decomposition. There is a large number of approaches to the decomposition methodologies (see, e.g., [5, 8, 13, 4]). The main difference and the main contribution of our methodology is that it was developed for such a system architecture, where we have already specified systems or components properties in a formal way and need to decompose this whole properties collection to a number of subcomponents to get readable and manageable specifications. Thus, the presented methodology allows us to decompose system or component architecture exactly on this point where we see that the component specification becomes too large and too complex. In many cases the real complexity of a component and, consequently, of its formal specification is realized only during the specification process, when we comes from semiformal (or, even harder, from informal) general description to a formal one – only by collecting and combining all the component properties together for the first time we also get the feeling of the component complexity for the first time. Moreover, during this step a number of component properties can added, in most cases some refinement is necessary.

In the context of hardware and software systems, the definition of (formal) *verification* is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property, using formal methods of mathematics, but we can also see a verification of a system as a special case of validation: if the property to prove is presented as an abstract specification, it remains to validate the system specification with respect to these abstract specification, i.e. to show that the *refinement relation* holds. We call this view *refinement-based verification*, and present according to these ideas an introduction to specification groups and refinement layers, as well as how these ideas of can be used to optimize the verification process, and which influences it has on the specification process.

The feasibility of this approach was proven on a number of case studies, the most interesting of them, Cruise Control System specification belonging to the Robert Bosch GmbH case study, can be seen at [11]: this system has 75 components (64 atomic components) and and yields approx. 17 KLOC of generated code and 38 KLOC of generated Isabelle/HOL theories, respectively.

2 Requirements: Refinement-Based Specification

Let a general specification S_0 of a system corresponds to the formalization of system requirements. In order to show that a concrete specification S_n that we get after n refinement steps fulfills the system requirements, we only need to show that the specification S_n is a refinement [3, 2] of the specification S_0. In this context, it is an important point what exactly a developer means by "refinement" on each refinement step (a behavioral refinement, an interface refinement, or a conditional refinement, changing time granularity etc.) and which specification semantics is used. We can see any proof about a system as the proof that a more concrete system specification is a refinement of a more abstract one: if the property to prove is presented as an abstract specification, it remains to validate the system specification with respect to these abstract specification, i.e. to show that the refinement relation holds (for details see [10]).

Fig. 1 represents the hierarchy in a *specification group* S in general. The number N of all specification in the group is larger or equal the number m of refinement layers: the specification S^1 is just a refinement specification of S, where S^j is a composition of specifications $S^j_1, \dots, S^j_n$ (where for the specifications $S^j_1, \dots, S^j_n$ the refinement layer j is the most abstract one) that builds a refinement of S^{j-1}.

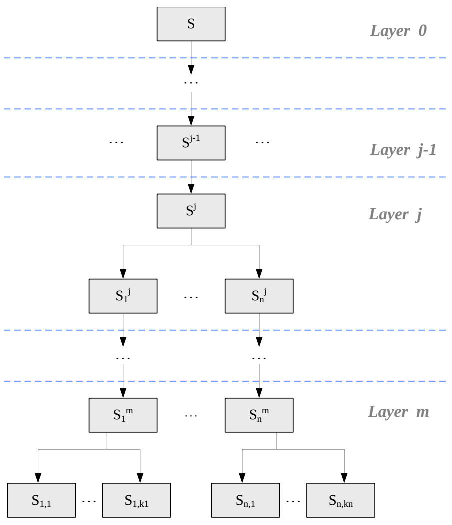

Figure 1: Refinement Layers of a Specification Group

Assuming a system S with corresponding list of requirements $L = [L_1, \dots, L_n]$:
$[\![S]\!] \Rightarrow [\![L]\!]$ where $[\![L]\!] = [\![L_1]\!] \wedge \dots \wedge [\![L_n]\!]$. For any new requirement R on the system S that we need to add to the list of its requirements L, $L \cup \{R\}$ (assuming R does not belong to the list of requirements) we can have the following cases that are intuitively clear.

(1) The system S has some requirement L_i that is less abstract than R:
$R \notin L \ \wedge \ \exists L_i \in L : L_i \Rightarrow R$.
We add R to the next level of abstraction L' (to the list with more abstract requirements,
$[\![L]\!] \Rightarrow [\![L']\!]$) using the same schema: $L' \cup \{R\}$.

(2) The list of requirements of the system S has a requirement that is more abstract than R:
$R \notin L \ \wedge \ \exists L_i \in L : R \Rightarrow L_i$.
We replace the requirement L_i in L by R, L_i will be added to the next level of abstraction L'. If S does not fulfill R, then S must be changed according to the new list of requirements.

(3) The system S has no requirements that are in some relation (more/less abstract) to R (R opens some new "dimension" of S):
$R \notin L \ \wedge \ \forall L_i \in L : \neg(L_i \Rightarrow R) \wedge \neg(R \Rightarrow L_i)$.

A list of requirements can also be represented by a formal specification. Thus, we allude the refinement layers. If the requirement specification can be extended, we always have a choice: either we extend the specification itself and don't make any changes of the refinement layers or we don't make any changes of the original specification, but add some new refinement layer with the extended version of the specification.

3 Architecture: Decomposition + Refinement

Let assume a formal specification of some component, which covers a large number of its properties, s.t. most of which have strong correlation, and let this component describes among others the system states and transitions between them, s.t. the resulting representation must correspond to a state transition diagram. If we specify this component as a single, non-composite, specification we get a set of formulas that is not really understandable. Trying to built a state transition diagram for the whole component, we will get a large automat with spaghetti-transitions between them – this representation will be useless and not manageable. Moreover, the later representation will be not fit the model checker restrictions. Therefore, we have a challenge to decompose it in a number of subcomponents to get some (more) readable specification. A simple, intuitive and informal, way to decompose a component is not suitable here. In this case we need to have some rules to decompose the component according to the kinds of its logical properties. Very important point here is to determine, whether the strong/weak causality property be preserved.

We start the decomposition to observe the properties that correspond to the different kinds of automats: Mealy and Moore. By definition, any state machine can be either a Mealy automat, where the output depends both on the current input and state, or a Moore automat, where the output depends only from the current state.

Generally, having a specification represented by a number of formulas, we can divide these formulas into two parts: formulas, which correspond to the definition of a Mealy automat, and formulas, which correspond to the definition of the Moore automat. Thus, having a component *CComp* describing large state machine, we can decompose it into two components, C and *CInf*, by this criteria.

As the next step we propose to use a decomposition schema for all local variables that have complicated computation specification: they are moved (together with the according specification parts that describe their computation) via decomposition from a component C to some extra component *CLoc*. This schema describes not only the way to write the specification *CLoc*, but also the changes we need to do in the specification C. Applying the decomposition schema we get two specifications, C' and $CLoc$, which composition results the specification C. After that we propose to use a decomposition schema for all output streams and corresponding formulas describing them (depending only on the component state, local variables and some inputs) that are moved via decomposition from a component C' to some extra component $COut$.

A special part of this is a discussion of the questions how to deal with assumptions about environment as well as parameters of a system/component. This questions correlate also with the discussion which kind of decomposition yields a system that is behavioral equal to the system before decomposition as well as which kind of decomposition yields a system that isn't equal to this system and is only a refinement of it.

Applying these decomposition methods allows us not only to keep the specification readable and manageable, but also find out a number of inconsistencies and underspecifications also during specification phase without starting a formal verification process, e.g. contradictory properties of the system behavior at some state or an underspecified behavior of a system.

Case Study:
These methods were successfully used within the case study on Cruise Control System that was motivated by Robert Bosch Gmbh. Technical report [11] presents the main methodological results. The formal decomposition schemas as well as the details of the case study do not be presented here for lack of space, now we give only a short overview of it.

We have to represent the system logic of the Cruise Control System by the component *LogicComp*, but its formal specification contains so much requirements that it is very hard to manage and to read – it covers a large number of properties most of which have a strong correlation. We decompose it into two components according the Mealy/Moore criteria. As result we get a very readable component *LogicInf* with simply 9 requirements inside that can be directly translated into the CASE tool representation.

The second component, *Logic*, is much more complicated - it contains now 48 formal requirements. That is manageable but only hardly readable (if we speaking about a formal and very precise specification of them). Thus, we decompose it again according to the local variables criteria. The obtained *LogicLoc* component is more readable and has only optimized 14 requirements and we can found the underspecifications in it, moreover, on this point we can make a standard parallel decomposition and represent a number of subcomponents by a number of instances of a single component.

After that we proceed in the same way. At the end, the specification *LogicMain* covers the main logic of the system and consists of 39 requirements – after correction of each underspecification case the overall number of requirements was increased, but at the end we get a collection of the specifications that are more complete, correct, readable, and manageable then the corresponding parts of starting component *LogicComp*. An abstract view on this architecture decomposition is presented on Figure 2.

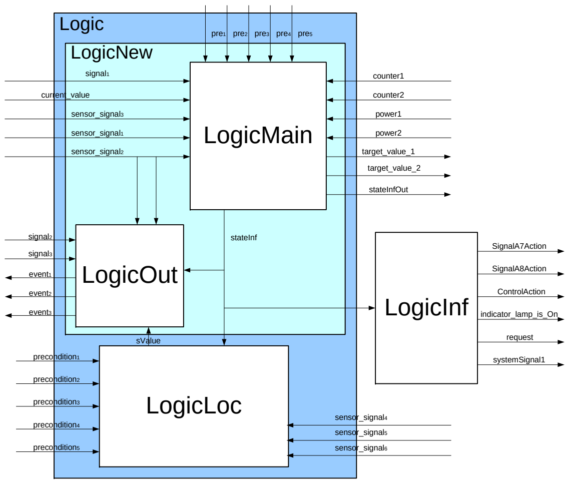

Figure 2: Decomposition of the *LogicComp*

4 FOCUS on Isabelle

The main ideas, presented in the paper, are language independent, but for the better readability and for better understanding of this ideas we shoe them ob the base of formal specifications presented in the Focus [3], a framework for formal specifications and development of interactive systems.[1] We can also see this methodology as an extension of the approach "Focus on Isabelle" [9] – it is integrated into a seamless development process[2], which covers both specification and verification, starts from informal specification and finishes by the corresponding verified C code. Given a system, represented in Focus, one can verify its properties by translating the specification to a Higher-Order Logic and subsequently using the theorem prover Isabelle/HOL or the point of disagreement can be found. For a detailed description of Isabelle/HOL see [7] and [12].

Focus is preferred here over other specification frameworks since it has an integrated notion of time and modeling techniques for unbounded networks, provides a number of specification techniques for distributed systems and concepts of refinement. For example, the B-method [1] is used in many publications on fault-tolerant systems, but it has neither graphical representations nor integrated notion of time. Moreover, the B-method also is slightly more low-level and more focused on the refinement to code rather than

[1]See `http://focus.in.tum.de`.
[2]See Verisoft-XT project, `http://www.verisoftxt.de`.

formal specification. Formal specifications of real-life systems can become very large and complex, and are as a result hard to read and to understand. Therefore, it is too complicated to start the specification process in some low-level framework directly. To avoid this problem FOCUS supports a graphical specification style based on tables and diagrams.

The main point in "FOCUS on Isabelle" is an alignment on the future proofs to make them simpler and appropriate for application not only in theory but also in practice. The proofs of some system properties can take considerable (human) time since the Isabelle/HOL is not fully automated. But considering "FOCUS on Isabelle" we can influence on the complexity of proofs already doing the specification of systems and their properties. Thus, the specification and verification/validation methodologies are treated as a single, joined, methodology with the main focus on the specification part.

In addition, the methodology helps to perform the next modeling step – translation to the case tool representation and deployment: we can schematically translate the FOCUS specification to a model in AutoFocus 3 (see also [6]), a tool for modeling and analyzing the structure and behavior of distributed, reactive, and timed computer-based systems.[3] Having such a model we can simulate it, prove its properties using model checking and using its translation to the theorem prover Isabelle/HOL, as well as we gan generate C code from it.

5 Conclusions

This paper presents the methodology for the system requirements and architecture w.r.t. their decomposition and refinement. The main contribution of our decomposition methodology is that it was developed for such a system architecture, where we know systems properties and need to decompose the whole properties collection to a number of subcomponent. Thus, the presented methodology allows us to decompose system or component architecture exactly on this point where we see that the component specification becomes too large and too complex to work with it. In addition, our methodology helps to perform the next modeling step – translation to the case tool representation and deployment.

This paper introduces also briefly the ideas of specification groups and refinement layers, as well as how the ideas of the refinement-based verification can be used to optimize the verification process, and which influences it has on the specification process. We can also see the presented methodology as an extension of the approach "FOCUS on Isabelle" [9] – it is integrated into a seamless development process, which covers both specification and verification, starts from informal specification and finishes by the corresponding verified C code.

[3]See `http://af3.in.tum.de`.

References

[1] J.-R. Abrial. *The B-book: assigning programs to meanings.* Camb.Univ.Press, 1996.

[2] M. Broy. Compositional refinement of interactive systems modelled by relations. *COMPOS'97: Revised Lectures from the International Symposium on Compositionality: The Significant Difference*, pages 130–149, 1998.

[3] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001.

[4] D. B. da Cruz and B. Penzenstadler. Designing, Documenting, and Evaluating Software Architecture. Technical Report TUM-I0818, TU München, 2008.

[5] C. Hofmann, E. Horn, W. Keller, K. Renzel, and M. Schmidt. The Field of Software Architecture. Technical Report TUM-I9641, TU München, 1996.

[6] F. Huber and B. Schätz. Integrated Development of Embedded Systems with AUTOFOCUS. Technical Report TUMI-0701, TU München, 2001.

[7] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[8] J. Philipps and B. Rumpe. Refinement of Pipe-and-Filter Architectures. In J. M. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods (FM'99)*, number LNCS 1708, pages 96 – 115. Springer, 1999.

[9] M. Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle.* PhD thesis, TU München, 2007.

[10] M. Spichkova. Refinement-based verification of interactive real-time systems. In *REFINE 2008.* ENTCS, 2008.

[11] M. Spichkova. Architecture: Methodology of Decomposition. Technical Report TUM-I1018, TU München, 2010.

[12] M. Wenzel. *The Isabelle/Isar Reference Manual.* TU München, 2004.

[13] D. Wild, A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, and S. Rittmann. An Architecture-Centric Approach towards the Construction of Dependable Automotive Software. In *Proc. of the SAE 2006 World Congress*, 2006.