Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 214 (2008) 131–157
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Refinement-Based Verification of Interactive Real-Time Systems

Maria Spichkova [1,2]

Institut für Informatik
Technische Universität München
Bolzmannstr. 3
D-85748 Garching, Germany

Abstract

Formal specification provides a system description that is much more precise than the natural language one and it can help to solve a lot of specification problems. But even a formal specification of a system can contain mistakes or can disagree with system's requirements. To cover this, we integrate a specification framework with a verification system. Given a system, represented in a formal specification framework Focus, one can verify its properties by translating the specification to a Higher-Order Logic and subsequently using the theorem prover Isabelle/HOL. Moreover, using this approach one can validate the refinement relation between two given systems. The approach uses the idea of refinement-based verification: we see any proof about a system as the proof that a more concrete system specification is a refinement of a more abstract one. The case when one needs to prove a single property of a system specification can also be seen as a refinement relation: this property can be defined as a Focus specification itself and then one needs just show that the system specification is its refinement. The major aspects of this approach are exemplified here by a case study on telematics (electronic data transmission) gateway.

Keywords: Formal Specification, Verification, Refinement, Real-Time Systems, Automotive Gateway

1 Motivation

The correctness of a system according to a given specification is essential, especially for safety-critical applications. Using formal methods we can not

[1] This work was partially funded by the German Federal Ministry of Education and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the author.

[2] Email: spichkov@in.tum.de

doi:10.1016/j.entcs.2008.06.007

only test correctness and safety, which is not enough for such kinds of interactive systems, but also prove them: verification guarantees fulfillment of the requirements. A formal specification is in general more precise than a natural language one, but it can also contain mistakes or disagree with requirements. Therefore, for safety critical systems 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 and is consistent.

We can treat any proof about a system as the proof that a more concrete system specification is a refinement of a more abstract one. The case when one needs to prove a single property of a system specification S can also be seen as a refinement relation: this property can be defined as a Focus specification S' itself and then one needs just show that the system specification S is a refinement of the specification S'. We call this view *refinement-based verification*. 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, where the definition of *validation* is the quality control and testing tasks and techniques used to determine if a work product (either an application or one of its components) conforms to its specified requirements, including operational, quality, interface, and design constraints. Thus, the verification means to proof properties of a system (more precisely, of a system specification) as some lemmas, where validation means to show that a more concrete specification fulfills all the properties of a more abstract one, i.e. that the refinement relation between these specifications holds. These concepts are very similar. Moreover, we can see 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.

In this paper we present the ideas of the refinement-based verification using a framework "Focus on Isabelle" [9]. Given system and requirements specifications, represented in a formal specification framework Focus, our method validates the refinement relation between them by translating the specifications to a Higher Order Logic and subsequent using the theorem prover Isabelle/HOL. In order to design systems in a step-wise, modular style we use Focus [5], a framework for formal specifications and development of interactive systems. 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 [2] 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 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, First-Order or Higher-Order Logic etc. directly. To avoid this problem Focus supports a graphical specification style based on tables and diagrams. In our approach we chose a prover for Higher-Order Logic, because the power of First-Order Logic is not enough to represent in a direct way several specifications of distributed interactive systems. As the verification system we have chosen Isabelle/HOL [8,11], an interactive semi-automatic theorem prover for Higher-Order Logic. The disadvantage of only semi-automated proofs is compensated by the advantage of using Higher-Order Logic.

The whole and detailed description of the framework "Focus on Isabelle" is presented in [9]. In this paper we show the application of its main contributions on the example of a verified specification of telematics (electronic data transmission) gateway for an automated emergency call. According to the proposal by the European Commission [1], such an automated emergency call should become mandatory in all new cars as of 2009. The gateway system itself is simple enough to be sketched in a few paragraphs, but it still possesses typical properties of such kind of systems and is very well suited for our method – its domain is safety-critical real-time applications, the system has a number of refinement layers. The specifications and their verification of the corresponding emergency call application and of the communication within a vehicle were introduced in [3] and in [7] respectively. Here we concentrate only on the gateway specification.

Outline. The rest of the paper is structured as follows: In Section 2 we introduce Focus and the representation of its major concepts in Isabelle/HOL. In Section 3 we discuss the ideas of the so-called refinement-based verification. In Section 4 we describe the application of represented ideas within a case study – a verified specification of the automotive gateway. Finally, in Section 5 we summarize the presented work.

2 FOCUS on Isabelle

This section provides a short introduction to Isabelle/HOL and to Focus, as well as the main points technique and methodology of translation from Focus specifications into ones in Isabelle/HOL.

A mapping of operators in Focus to the corresponding definitions in HOL alone is not sufficient for the method to become easy. Because of this, we also need a specification and proof methodology. The main point in our methodology is an alignment on the future proofs to make them simpler and appropriate for application not only in theory but also in practice. For this we have performed a number of case studies, whose results have helped us to find out different problem points [3] and corresponding solutions for the coupling FOCUS and Isabelle/HOL. The proofs of some system properties can take considerable (human) time since the Isabelle/HOL is not fully automated. But considering our framework "FOCUS on Isabelle" [9] we can influence on the complexity of proofs already doing the specification of systems and their properties, e.g. modifying (reformulating) specification to simplify the Isabelle/HOL proofs for a translated FOCUS specification. Thus, the specification and verification/validation methodologies are treated as a single, joined, methodology with the main focus on the specification part. For this purpose we introduce a number of additional FOCUS operators [4]. At the most cases we advise to specify the input/output relation on the streams based on time intervals. Argumentation over time intervals of streams help us have not only more clear and readable FOCUS specifications, but also simpler and shorter proofs in Isabelle/HOL.

2.1 FOCUS and Isabelle/HOL

A distributed system in FOCUS is represented by its *components* [5]. Components that are connected by communication lines called *channels*, can interact or work independently of each other. The channels in FOCUS are *asynchronous communication links* without delays. They are directed, reliable, and order preserving. Via these channels components exchange information in terms of *messages* of specified types. The formal meaning of a FOCUS specification is a relation between the *communication histories* for the external input and output channels. The specifications can be structured into a number of formulas each characterizing a different kind of property, the most prominent classes of them are *safety* and *liveness properties.* FOCUS supports a variety of *specification styles* which describe system components by logical formulas or by diagrams and tables representing logical formulas.

Isabelle [8] is implemented in the functional programming language ML. The base types in Isabelle/HOL are *bool*, the type of truth values and *nat*, the type of natural numbers. The base type constructors are *list*, the type of lists,

[3] Like representation of mutually recursive functions, specification replications, sheaves of channels, a large number of refinement layers, etc.

[4] In this paper we discuss only these FOCUS operators, which are used in the case study.

[5] A component in FOCUS means a "logical component" and not a physical one.

and *set*, the type of sets. Function types are denoted by ⇒. The type variables are denoted by *'a*, *'b* etc. Terms in Isabelle/HOL are formed as in functional programming by applying functions to arguments. Terms may also contain λ-abstractions. To specify a system with Isabelle means creating *theories*, which are named collection of types, functions (constants), and theorems (lemmas). For a detailed description of Isabelle/HOL see [8] and [11].

2.2 Concept of Streams

The central concept in Focus are *streams*, that represent communication histories of *directed channels*. Streams in Focus are functions mapping the indexes in their domains to their messages. For any set of messages M, M^ω denotes the set of all streams, M^∞ and M^* denote the sets of all infinite and all finite streams respectively. $M^{\underline{\omega}}$ denotes the set of all timed streams, $M^{\underline{\infty}}$ and $M^{\underline{*}}$ denote the sets of all infinite and all finite timed streams respectively. A *timed stream* is represented by a sequence of messages and *time ticks*, the messages are also listed in their order of transmission. The ticks model a discrete notion of time.

The timed domain is the most important one for representation of distributed systems with real-time requirements. Specifications of embedded systems must be *timed*, because by representing a real-time system as an untimed specification a number of properties of the system are loosed (e.g. the causality property) that are not only very important for the system, but also help us to make proofs easier. The definition in Isabelle/HOL of the Focus stream types is given below. Another ways of streams formalizations as well as the related work for the approach "Focus on Isabelle" are discussed in [9].

- *Finite untimed streams* of type *'a* are represented by the list type: *'a list*. This type will be used to argue about a sequence of messages that are transmitted during a time unit.
- *Finite timed streams* of type *'a* are represented by the type *'a list list*, which will be used to argue about a timed stream that was truncated at some point of time – each list here represents a sequences of messages that are transmitted during the corresponding time unit.
- *Infinite untimed streams*, *nat* ⇒ *'a*, will be used to represent in Isabelle/HOL the local variables from Focus specifications.
- *Infinite timed streams* of type *'a* are represented by the type *'a istream* that represents the functional type *nat* ⇒ *'a list*. For specifying the input and the output streams to represent the behavior of an embedded system only this kind of streams is used – timed streams must be infinite because time never halts.

To simplify the specification of the real-time systems we introduce an additional FOCUS operator $\mathsf{ti}(s, n)$ that yields the list of messages that are in the timed stream s between the ticks $n-1$ and n (at the nth time unit). According to our representation of the timed FOCUS streams this operator corresponds in Isabelle/HOL simply to $s\ n$.

The predicate ts holds for a timed stream s, iff s is time-synchronous in the sense that exactly one message is transmitted in each time interval.

We define an Isabelle/HOL predicate $msg\ n\ s$ that is equal modulo syntax to the FOCUS operator $\mathsf{msg}_n(s)$, which holds for a timed stream s, if this stream contains at every time unit at most n messages.

2.3 Specifications and the Concept of Refinement

FOCUS specifications can be *elementary* or *composite*. Any elementary FOCUS specification has the following syntax:

Name (Parameter_Declarations) ═══ Frame_Labels

in *Input_Declarations*

out *Output_Declarations*

Body

Name is the name of the specification; *Frame_Labels* lists a number of frame labels, e.g. *untimed*, *timed* or *time-synchronous*, that correspond to the stream types in the specification (see Sect. 2.2); *Parameter_Declarations* lists a number of parameters (optional); *Input_Declarations* and *Output_Declarations* list the declarations of input and output channels respectively. *Body* characterizes the relation between the input and output streams, and can be a number of formulas, or a table, or diagram or a combination of them.

Definition 2.1 For any elementary timed parameterized specification S we define its semantics, written $[\![S]\!]$, to be the formula:

$$i_S \in I_S^{\underline{\infty}} \ \wedge\ p_S \in P_S \ \wedge\ o_S \in O_S^{\underline{\infty}} \ \wedge\ B_S \tag{1}$$

where i_S and o_S denote lists of input and output channel identifiers, I_S and O_S denote their corresponding types, p_S denotes the list of parameters and P_S denotes their types, B_S is a formula in predicate logic that describes the body of the specification S. $\square$

To define the semantics of a timed specification in Isabelle/HOL we introduce first tree predicates, $\mathsf{inStream}$, $\mathsf{outStream}$, and $\mathsf{locStream}$ over infinite timed streams. The predicate $\mathsf{inStream}/\mathsf{outStream}/\mathsf{locStream}$ is true, if the channel identifier corresponds to an input/output/local stream. Now we can define the semantics of an elementary timed specification with the input channels $i_1, \dots, i_n$ and the output channels $o_1, \dots, o_m$ (and with parameters $p_1, \dots, p_k$) in Isabelle/HOL in the same way as it is defined in Focus:

$$\bigwedge_{1}^{n} \mathsf{inStream}(i_j) \ \wedge \ \bigwedge_{1}^{m} \mathsf{outStream}(o_j) \ \wedge \ \mathit{body} \tag{2}$$

where the Isabelle/HOL predicate *body* describes here the relation (with k extra parameters) between the input and output streams and is equal modulo syntax to B_S that is conjunction of all propositions in the body of the specification S). The order of the parameters in the relation must be the following one: number of channels in the sheaf, input streams, specification parameters, output streams. For the proofs of the properties we need only the predicate *body*. Therefore, only this part will be denoted later as semantic of the specification. The conjunction of the predicates $\mathsf{inStream}/\mathsf{outStream}/\mathsf{locStream}$ will be defined in Isabelle/HOL separately, because this part will be used only to show that the correctness of syntactic interface. [6]

Composite specifications are built hierarchically from elementary ones using constructors for composition and network description and can be represented in the *graphical*, the *constraint* and *operator* style. Semantics of a composite Focus specification is defined in [5] as follows.

Definition 2.2 For any composite specification S consisting of n subspecifications $S_1, \dots, S_n$, we define its semantics, written $[\![S]\!]$, to be the formula:

$$[\![S]\!] \overset{def}{=} \exists\, l_S \in L_S^{\infty} : \bigwedge_{j=1}^{n} [\![S_j]\!] \tag{3}$$

where l_S denotes a list of local channel identifiers and L_S denotes their corresponding types. □

To simplify the Isabelle/HOL proofs we split the specification semantics into 3 parts (see Equation 2): input and output streams, as well as the semantic of the specification body. The semantics of the body of a composite specification S in Isabelle/HOL is a predicate

[6] The signature of the corresponding predicate will be equal to the signature of the predicate *body*.

$$\exists\, l_S \in istream_S : \bigwedge_{j=1}^{n} predicateS_j \tag{4}$$

where l_S denotes a list of local channel identifiers and $istream_S$ denotes their corresponding types, and $predicateS_j$ denotes the predicate that is a representation in Isabelle/HOL of the Focus specification S_j. The representation of the parts responsible for syntactic interface is done in the similar way.

To prove that a system specification fulfills its requirements the idea of refinement can be used. In Focus we can have a general specification S_0 of a system that corresponds to the formalization of system requirements. To show that a concrete specification S_n, which we get after n refinement steps, fulfills the system requirements, we only need to show that the specification S_n is a refinement of the specification S_0.

The Focus specification framework uses three basic refinement relations:

Behavioral Refinement:

The related specifications S_1 and S_2 must have the same syntactic interface. The refined specification S_2 may meet further requirements in addition to the requirements on the more abstract specification S_1. At the same time, the more concrete specification S_2 must meet all the requirements on the specification S_1. This kind of refinement is used to reduce the number of possible output histories for a given input history.

Interface Refinement:

This kind of refinement is a generalization of the behavioral refinement, it allows to work on the different levels of interface abstraction: the related specifications S_1 and S_2 may have different syntactic interface.

Conditional Refinement:

The conditional refinement is a generalization of the interface refinement, it allows the introduction of additional input assumptions.

We are using here the definitions of the behavioral refinement from [5].

Definition 2.3 A specification S_2 is called a behavioral refinement ($S_1 \leadsto S_2$) of a specification S_1 if they have the same syntactic interface and any I/O history of S_2 is also an I/O history of S_1. □

Therefore, in order to show that our concrete specification S_2 fulfills the system requirements S_1, we only need to show that $[\![S_2]\!] \Rightarrow [\![S_1]\!]$, i.e. the relation $\leadsto$ of behavioral refinement is defined by equivalence

$$(S_1 \leadsto S_2) \Longleftrightarrow ([\![S_2]\!] \Rightarrow [\![S_1]\!]) \tag{5}$$

Formally, we need to show that any I/O history of S_2 is an I/O history of S_1, but S_1 may have additional I/O histories. In Isabelle it means to prove that the formula that corresponds to the semantics of the specification body $[\![S_2]\!]$ implies the formula that corresponds to $[\![S_1]\!]$. This definition of refinement does not exclude that the set of I/O histories of S_2 can be empty. It means $[\![S_2]\!]$ is false[7] and the refinement relation is true. Thus, this point must be also taken into account proving the refinement relation in Isabelle/HOL. The verification of the property that both specifications have the same syntactic interface will be done separately.

3 Refinement-Based Verification

In Focus we can have a general specification S_0 of a system that corresponds to the formalization of system requirements. Therefore, in order to show that our 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 [5,4,6] 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.

In this section we introduce first of all the representation of the refinement layers of a specification group and the general ways of their representation in Isabelle/HOL. After that we discuss the representation in Isabelle/HOL of behavioral refinement[8] and how the ideas of the refinement-based verification can be used.

3.1 Refinement Layers of a Specification Group

Figure 1 represents the hierarchy in a specification group S in general. It has m refinement layers:

- specification S^1 is a refinement specification of S,
- $\dots$,
- 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}.
- $\dots$,

[7] This can happen if the specification S_2 is inconsistent.

[8] For the details of representation in Isabelle/HOL of interface and conditional refinement we refer to the [9].

- S_i^m, $1 \le i \le n$ is a composition of specifications $S_{1,k1}^m, \dots, S_{n,kn}^m$ that builds a refinement of S^{m-1}, where the specifications $S_{1,k1}^m, \dots, S_{n,kn}^m$ are elementary ones.

The number N of all specification in the group is larger or equal[9] to the number of layers.

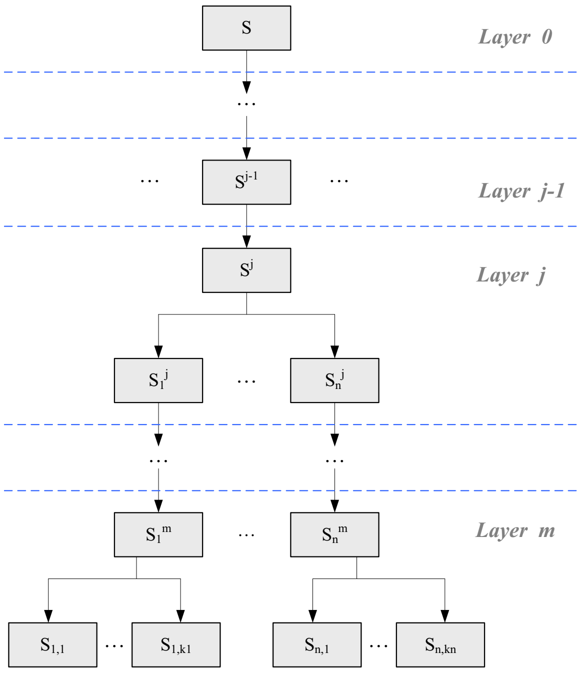


Fig. 1. Refinement Layers of a Specification Group S

3.2 Behavioral Refinement

According to the definition of behavioral refinement (Definition 2.3), in order to show that the more concrete specification S_2 (e.g. a specification of a system architecture) fulfills the more abstract S_1 (e.g. system requirements), we only need to show

$$[\![S_2]\!] \Rightarrow [\![S_1]\!] \tag{6}$$

In Isabelle it means to prove that the formula that corresponds to $[\![S_2]\!]$ implies the formula that corresponds to $[\![S_1]\!]$.

Definitions 2.1 and 2.2 (semantics of an elementary and a composite specification respectively) imply that the semantics of any Focus specification S

[9] Equality is possible only in the case, when we do not have any compositional specification in the group.

can be represented by

$$i_S \in I_S^{\infty} \ \wedge \ o_S \in O_S^{\infty} \ \wedge \ B_S \tag{7}$$

where i_S and o_S denote sets of input and output channel identifiers, I_S and O_S denote their corresponding types, and B_S is a logic formula. In the case of composite specification consisting of n subspecifications $S_1, \dots, S_n$:

- The formula $[\![B_S]\!]$ is equal to formula

  $$\exists\, l_S \in L_S^{\infty} : \bigwedge_{j=1}^{n} [\![B_{S_j}]\!]$$

  where l_S denotes a list of local channel identifiers and L_S denotes their corresponding types, and B_{S_j} denotes the logic formula describing the body of the specification S_j.
- The lists of input and output channel identifiers, i_S and o_S, is a concatenation of corresponding lists of specifications $S_1, \dots, S_n$, except those that are used as local channels (belong to l_S).

Together with Equation 5, this implies that the formal definition of the behavioral refinement [5] allows that the refined specification may have more input and output channels in addition to the input and output channels of the abstract specification. Let the specification S_2 be a behavioral refinement of the specification S_1 in the meaning of Definition 2.3: $S_1 \rightsquigarrow S_2$. According to the Equation 7 the semantics of these specification can be represented as follows:

$$[\![S_1]\!] = i_{S_1} \in I_{S_1}^{\infty} \ \wedge \ o_{S_1} \in O_{S_1}^{\infty} \ \wedge \ B_{S_1}$$
$$[\![S_2]\!] = i_{S_2} \in I_{S_2}^{\infty} \ \wedge \ o_{S_2} \in O_{S_2}^{\infty} \ \wedge \ B_{S_2}$$

Then, according to the Equation 5 we can conclude the following:

$$\begin{aligned}
&(S_1 \rightsquigarrow S_2) \iff \\
&([\![S_2]\!] \ \Rightarrow \ [\![S_1]\!]) \ \Leftrightarrow \\
&(i_{S_2} \in I_{S_2}^{\infty} \ \wedge \ o_{S_2} \in O_{S_2}^{\infty} \ \wedge \ B_{S_2}) \ \Rightarrow \ (i_{S_1} \in I_{S_1}^{\infty} \ \wedge \ o_{S_1} \in O_{S_1}^{\infty} \ \wedge \ B_{S_1}) \\
&\Leftrightarrow \\
&(i_{S_2} \in I_{S_2}^{\infty} \Rightarrow i_{S_1} \in I_{S_1}^{\infty}) \ \wedge \ (o_{S_2} \in O_{S_2}^{\infty} \Rightarrow o_{S_1} \in O_{S_1}^{\infty}) \ \wedge \ (B_{S_2} \Rightarrow B_{S_1})
\end{aligned}$$

The conjunct $i_{S_2} \in I_{S_2}^{\infty} \Rightarrow i_{S_1} \in I_{S_1}^{\infty}$ means that the set of input channels of the specification S_1 is a subset of the set of input channels of the specification S_2: $i_{S_2} \subseteq i_{S_1}$, and the conjunct $o_{S_2} \in O_{S_2}^{\infty} \Rightarrow o_{S_1} \in O_{S_1}^{\infty}$ means that the set of output channels of the specification S_1 is a subset of the set of output channels of the specification S_2: $o_{S_2} \subseteq o_{S_1}$.

For the cases when the more strict version of behavioral refinement is needed, where both specifications (an abstract one and a refined one) must have exactly the same syntactic interface. Thus, we introduce a new definition of the behavioral refinement – the *strict behavioral refinement.*

Definition 3.1 A specification S_2 is called a *strict behavioral refinement* ($S_1 \rightsquigarrow S_2$) of a specification S_1 if

- they have *exactly the same syntactic interface* and
- any I/O history of S_2 is also an I/O history of S_1.

We define the relation $\rightsquigarrow$ of strict behavioral refinement by equivalence

$$\begin{aligned}&(S_1 \rightsquigarrow S_2)\\ &\Leftrightarrow \qquad (8)\\ &((i_{S_1} \in I_{S_1}^{\infty} = i_{S_2} \in I_{S_2}^{\infty}) \ \wedge \ (o_{S_1} \in O_{S_1}^{\infty} = o_{S_2} \in O_{S_2}^{\infty}) \ \wedge \ (B_{S_2} \Rightarrow B_{S_1}))\end{aligned}$$

where i_{S_1} and i_{S_2} (o_{S_1} and o_{S_2}) denote lists of input (output) channel identifiers of the specifications S_1 and S_2 respectively, I_{S_1}, I_{S_2}, O_{S_1} and O_{S_2} denote their corresponding types, B_{S_1} and B_{S_2} are logic formulas in terms of the Equation 7. □

The definition of refinement does not exclude that the set of I/O histories of S_2 is empty. It means $[\![S_2]\!]$ is false and the refinement relation is true. This can happen if the specification S_2 is inconsistent.

3.3 Verification

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 example, the specification S fulfills the property P_1 and P_2 under assumption that the properties A_1, A_2 and A_3 hold. Thus, the specification S can be seen as

- a (behavioral) refinement of the specification S' that is combination of assumptions A_1, A_2, A_3 and guarantee P_1.
- a (behavioral) refinement of the specification S'' that is combination of assumption assumptions A_1, A_2, A_3 and guarantee P_2.

In most cases a system must be verified with respect not to a single property, but with respect to a number of properties. In this case it is more sufficient to specify these properties as a single specification to exclude possible inconsistencies. Therefore, we verify that the specification S^R is a (behavioral) refinement of the specification S that is combination of assumption A_1, A_2, A_3 and guarantee $P_1 \ \wedge \ P_2$.

Let S be some specification of a system and let a specification L consist of properties $L_1, \dots, L_n$ of this system. The corresponding refinement lemma looks like $[\![S]\!] \Rightarrow [\![L]\!]$. Applying the definition of the Isabelle/HOL predicate which corresponds to $[\![L]\!]$ we get

$$[\![S]\!] \Rightarrow ([\![L_1]\!] \ \wedge \dots \wedge \ [\![L_n]\!])$$

Then we can split the verification goal into n subgoals, each of them can be proved as a separate lemma:

$$[\![S]\!] \Rightarrow [\![L_1]\!]$$
$$\dots$$
$$[\![S]\!] \Rightarrow [\![L_n]\!]$$

In some cases the requirements (properties) can be sorted to get a nested hierarchy. Assuming e.g. the following two requirements, namely L_1 and L_2, of some system S_1 that has an output channel y of type $\mathbb{N}$:

$$\forall\, t \in \mathbb{N} : \ \mathsf{ti}(y, t) \neq \langle\rangle \tag{9}$$

and

$$\forall\, t \in \mathbb{N} : \ \#\mathsf{ti}(y, t) = 2 \tag{10}$$

The second requirement, L_2, is a refinement of the first one, L_1. Therefore, if we show that the system S_1 fulfills the second requirement, we do not need to show that it fulfills the first requirement, but we need to show that the refinement relation between L_1 and L_2 holds, which in most cases is easer than to show that S_1 fulfills L_1.

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.

(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: $L' \cup \{R\}$, see Figure 2 (a).

(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' (to the list with more abstract requirements), see Figure 2 (b). 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):
$R \notin L \ \wedge \ \forall\, L_i \in L : \neg(L_i \Rightarrow R) \wedge \neg(R \Rightarrow L_i)$.
For example, assuming the properties L_1 (see Equation 9) and L_2 (see Equation 10). The property R, which says that every first element [10] of the time interval $t + 2$ ($\forall\, t \in \mathbb{N}$) must be equal to the natural number 5

$$\forall\, t \in \mathbb{N} : \ \mathsf{ft.ti}(y, t + 2) = 5,$$

does not imply L_1, because it is only about time intervals 2, 4, 6, 8 etc., and it also does not imply L_2, because it says nothing about the length of message list at the time intervals. Neither L_1 nor L_2 imply R, because they say nothing about the message values.

The R will be added to the list of requirements L, see Figure 2 (c). If S does not fulfill R, then S must be changed according the new list of requirements.

The lists of requirements are specifications itself. Thus, we allude the refinement layers (see Figure 1). 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

[10] The first element of a message sequence s is denoted by $\mathsf{ft}.s$.

- or we don't make any changes of the original specification, but add some new refinement layer with the extended version of the specification.

See Section 4.7 for an example.

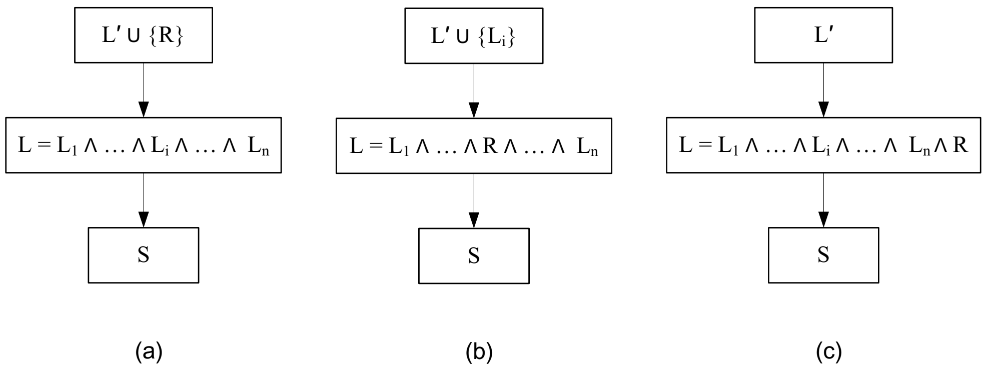


Fig. 2. Adding new requirement to the list of requirements of the specification

The legitimate question is, where we need to argue about such more abstract requirement like L_1 at all having more precise requirements like L_2, and why we cannot just remove them. The point is, that a number of system requirements comes out from the argumentation about interaction with another components or systems. Considering a system S_3 that consist of two subsystems: the system S_1 and some system S_2, and let the channel y be a local one for the system S_3, i.e. this output channel channel of S_1 will be an input channel for S_2. Thus, all assumptions of S_2 about this input channel y must be fulfilled by S_1 as its new requirements.

Assuming we have composite specifications $A, \dots, Z$, their subcomponents and the corresponding requirements specifications $AReq, \dots, ZReq$. If the refinement relations $AReq \rightsquigarrow A$, …, $ZReq \rightsquigarrow Z$ have been proved, these requirements can be used to prove properties of a specification S, which is composed of $A, \dots, Z$ – in most cases it is easier to prove the main lemma using these requirements specification, than to use the architecture specifications directly (see Section 4.7 for examples).

4 Automotive-Gateway

This section introduces the case study on telematics (electronic data transmission) gateway that was done for the Verisoft project [10]. If the gateway receives from a ECall application of a vehicle a signal about crash (more precise, the command to initiate the call to the Emergency Service Center), and after the establishing the connection it receives the command to send the crash data (these data were already received and stored in the internal buffer of the gateway), these data will be resent to the Emergency Service Center and the voice communication will be established, assuming that there is no connection fails.

In this section we discuss at first the Focus specifications of the gateway system and its requirements. The specifications are subsequently schematically translated into Isabelle/HOL using the representation of Focus streams presented above. After that the proof of the refinement lemmas is discussed. The refinement lemma says that the gateway architecture specification fulfills its requirements. Since the overall representation of the gateway system in Focus and Isabelle/HOL as well as the proofs of auxiliary lemmas are too extensive for this paper, we describe here only some aspect of the specifications and proofs, and show only a simple and short parts of the specifications to give a feeling how the approach works. For the technical details of the case study we would like to refer to [9].

4.1 Representation of Datatypes

The datatype *ECall_Info* represents a tuple, consisting of the data that the Emergency Service Center needs – here we specify these data to contain the vehicle coordinates and the collision speed, they can also extend by some other information. The datatype *GatewayStatus* represents the status (internal state) of the gateway.

$$\begin{array}{lcl}
\mathsf{type}\ \mathit{ECall\_Info} & = & \mathit{ecall}(\mathit{coord} \in \mathit{Coordinates}, \mathit{speed} \in \mathit{CollisionSpeed}) \\
\mathsf{type}\ \mathit{GatewayStatus} & = & \{\ \mathit{init\_state},\ \mathit{call},\ \mathit{connection\_ok}, \\
 & & \quad \mathit{sending\_data},\ \mathit{voice\_com}\ \}
\end{array}$$

The Isabelle/HOL specifications of these types are equal modulo syntax to the corresponding types in the Focus specification. To specify the automotive gateway we will use a number of datatypes consisting of one or two elements: $\{sc\_ack\}$, $\{init, send\}$, $\{vc\_com\}$ and $\{stop\_vc\}$. We name these types *aType*, *reqType*, *vcType* and *stopType* correspondingly, and represent them in Isabelle/HOL schematically.

4.2 Gateway System: Architecture and Requirements

The specification *GatewaySystemReq* specifies the requirements on gateway system: assuming that the input streams *req* and *stop* can contain at every time interval at most one message, and assuming that the stream *lose* contains at every time interval exactly one message. The stream *lose* represents the connection status: the message true at the time interval t corresponds to the connection failure at this time interval, the message false at the time interval t means that at this time interval no data loss on the gateway connection.

GatewaySystemReq(const d $\in \mathbb{N}$) — timed

in $req : \{init, send\};\ dt : ECall\_Info;$
$stop : stop\_vc;\ \ lose : Bool$

out $ack : GatewayStatus;\ \ vc : \{vc\_com\}$

asm $\mathsf{msg}_1(req)\ \wedge\ \mathsf{msg}_1(stop)\ \wedge\ \mathsf{ts}(lose)$

gar

$$\begin{array}{l}
\forall\, t, k \in \mathbb{N} : \\
\quad \mathsf{ti}(ack, t) = \langle init\_state \rangle\ \wedge\ \mathsf{ti}(req, t+1) = \langle init \rangle \\
\quad \wedge\ \mathsf{ti}(req, t+2) = \langle\rangle \\
\quad \wedge\ (\forall\, t_1 \in \mathbb{N} :\ t_1 \le t \to \mathsf{ti}(req, t1) = \langle\rangle) \\
\quad \wedge\ (\forall\, m \in \mathbb{N} :\ m \le k+3 \to \mathsf{ti}(req, t+m) \ne \langle send \rangle) \\
\quad \wedge\ \mathsf{ti}(req, t+3+k) = \langle send \rangle \wedge \mathsf{last}^{ti}(dt, t+2) \ne \langle\rangle \\
\quad \wedge\ (\forall\, j \in \mathbb{N} :\ j \le (4+k+d+d) \to \mathsf{ti}(lose, t+j) = \langle \mathsf{false} \rangle) \\
\quad \to \\
\quad \mathsf{ti}(vc, t+4+k+d+d) = \langle vc\_com \rangle
\end{array}$$

If

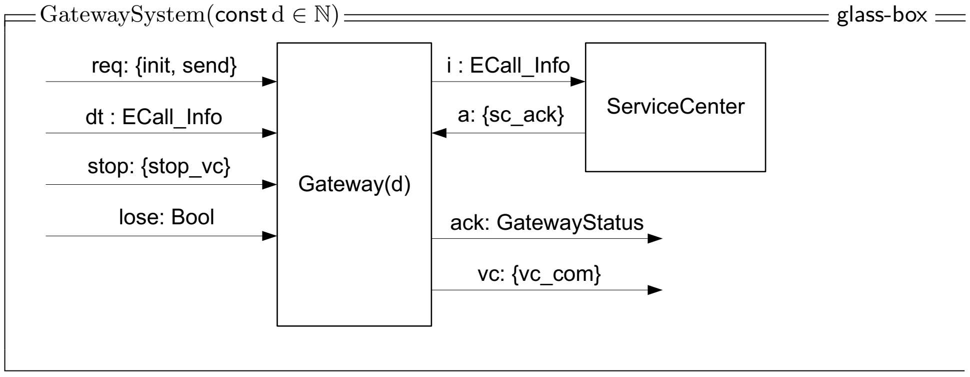
- at any time interval t the gateway system is in the initial state, $\mathsf{ti}(ack, t) = \langle init\_state \rangle$, and
- at time interval $t+1$ the signal about crash comes at first time (more precise, the command to initiate the call to the Emergency Service Center), $\mathsf{ti}(req, t+1) = \langle init \rangle\ \wedge\ (\forall\, t_1 \in \mathbb{N} :\ t_1 \le t \to \mathsf{ti}(req, t1) = \langle\rangle)$, and
- after $3+m$ time intervals the command to send the crash data comes at first time $(\forall\, m \in \mathbb{N} :\ m \le k+3 \to \mathsf{ti}(req, t+m) \ne \langle send \rangle)\ \wedge\ \mathsf{ti}(req, t+3+k) = \langle send \rangle$, and
- the gateway system has received until the time interval $t+2$ the crash

data,
$\mathsf{last}^{ti}(dt, t+2) \neq \langle\rangle$, and

- there is no connection fails from the time interval t until the time interval $t+4+k+2d$,
  $\forall j \in \mathbb{N} : \ j \leq (4+k+d+d) \rightarrow \mathsf{ti}(lose, t+j) = \langle \mathsf{false} \rangle$

then at time interval $t+4+k+2d$ the voice communication is established, $\mathsf{ti}(vc, t+4+k+d+d) = \langle vc\_com \rangle$.

The Focus specification of the gateway system and the Focus representation of the specification *GatewaySystem* as plain text are presented below:



GatewaySystem(const d $\in \mathbb{N}$) ═══ timed

in	$req : \{init, send\}$; $dt : ECall\_Info$; $stop : stop\_vc$; $lose : Bool$
out	$ack : GatewayStatus$; $vc : \{vc\_com\}$
loc	$a : \{sc\_ack\}$; $i : ECall\_Info$

$(ack, i, vc) := Gateway(d)(req, dt, a, stop, lose)$

$(a) := ServiceCenter(i)$

On the example of these two specifications we show the results of the specification translation from Focus to Isabelle/HOL.[11] We convert the Focus specifications *GatewaySystem* and *GatewaySystemReq* into Isabelle/HOL predicates *GatewaySystemReq* and *GatewaySystem* respectively. This translation is done schematically, according to the approach "Focus on Isabelle".

[11] For the whole version of the translation as well as the technical details we would like to refer to [9].

```
constdefs
  GatewaySystem ::
   reqType istream ⇒ ECall_Info istream ⇒
    stopType istream ⇒ bool istream ⇒ nat ⇒
    GatewayStatus istream ⇒ vcType istream ⇒ bool

 GatewaySystem req dt stop lose d ack vc
  ≡ ∃ a i. (Gateway req dt a stop lose d ack i vc) ∧ (ServiceCenter i a)

constdefs
  GatewaySystemReq ::
   reqType istream ⇒ ECall_Info istream ⇒
    stopType istream ⇒ bool istream ⇒ nat ⇒
    GatewayStatus istream ⇒ vcType istream ⇒ bool

 GatewaySystemReq req dt  stop lose d ack vc
  ≡
 ((msg (1::nat) req) ∧ (msg (1::nat) stop) ∧ (ts lose))
   ⟶
  (∀ (t::nat) (k::nat).
  ( ack t = [init_state] ∧ req (Suc t) = [init]
  ∧ (∀ t1. t1 ≤ t ⟶ req t1 = [])  ∧ req (t+2) = []
  ∧ (∀ m. m < k + 3 ⟶ req (t + m) ≠ [send])
  ∧  req (t+3+k)  = [send] ∧  inf_last_ti dt (t+2) ≠ []
  ∧ (∀ (j::nat).
     j ≤ (4 + k + d + d) ⟶ lose (t+j) = [False])
  ⟶ vc (t + 4 + k + d + d) = [vc_com]) )
```

4.3 ECall Service Center

The component *ServiceCenter* represents the behavior of the Emergency (ECall) Service Center from the gateway point of view: if at time t a message about a vehicle crash comes, it acknowledges this event by sending the at time $t+1$ message *sc_ack* that represents the attempt to establish the voice communication with the driver or a passenger of the vehicle (*voice_communication* output message of the *Gateway* component) – if there is no connection failure, after d time intervals the voice communication will be started.

4.4 Gateway: Requirements Specification

We define the formal specification of the gateway requirement, presented in the previous section, as Focus specification *GatewayReq*:

(i) If at time t the gateway is in the initial state *init_state*, and it gets the command to establish the connection with the central station, and also there is no environment connection problems during the next 2 time intervals, it establishes the connection at the time interval $t+2$, $\mathsf{ti}(ack, t+2) = \langle connection\_ok \rangle$.

(ii) If at time t the gateway has establish the connection, $\mathsf{ti}(ack, t) = \langle connection\_ok \rangle$, and it gets the command to send the E-Call data to the central station ($\mathsf{ti}(req, t+1) = \langle send \rangle$), and also there is no environment connection problems during the next $d+1$ time intervals, $\forall\, k \in \mathbb{N} :\ k \leq d+1 \rightarrow \mathsf{ti}(lose, t+k) = \langle \mathsf{false} \rangle$, then it sends the last corresponding data.[12] The central station becomes these date at the time $t+d$.

(iii) If the gateway becomes the acknowledgment from the central station that it has receives the sent E-Call data, and also there is no environment connection problems, then the voice communication is started.

GatewayReq(const d $\in \mathbb{N}$) — timed

in	$req : \{init\_connect\};\ dt : ECall\_Info;\ \ a : \{sc\_ack\};$ $stop : stop\_vc;\ \ lose : Bool$
out	$ack : GatewayStatus;\ \ i : ECall\_Info;\ \ vc : \{voice\_com\}$

asm $\mathsf{msg}_1(req) \ \wedge\ \mathsf{msg}_1(a) \ \wedge\ \mathsf{msg}_1(stop) \ \wedge\ \mathsf{ts}(lose)$

gar

$\forall\, t \in \mathbb{N} :$

$$\begin{aligned}
&\mathsf{ti}(ack, t) = \langle init\_state \rangle \ \wedge\ \mathsf{ti}(req, t+1) = \langle init \rangle \\
&\wedge\ \mathsf{ti}(lose, t+1) = \langle \mathsf{false} \rangle \ \wedge\ \mathsf{ti}(lose, t+2) = \langle \mathsf{false} \rangle \\
&\qquad \rightarrow \mathsf{ti}(ack, t+2) = \langle connection\_ok \rangle
\end{aligned}$$

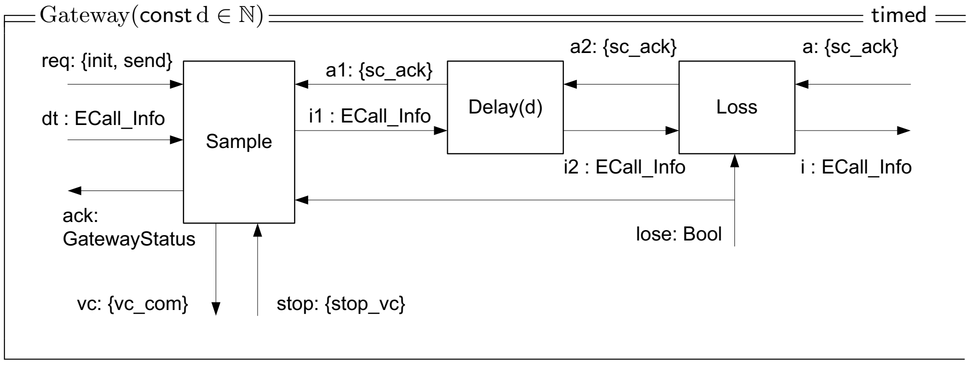
$$\begin{aligned}
&\mathsf{ti}(ack, t) = \langle connection\_ok \rangle \ \wedge\ \mathsf{ti}(req, t+1) = \langle send \rangle \\
&\wedge\ (\forall\, k \in \mathbb{N} :\ k \leq d+1 \rightarrow \mathsf{ti}(lose, t+k) = \langle \mathsf{false} \rangle) \\
&\qquad \rightarrow \mathsf{ti}(i, t+d+1) = \mathsf{last}^{ti}(dt, t) \ \wedge\ \mathsf{ti}(ack, t+1) = \langle sending\_data \rangle
\end{aligned}$$

$$\begin{aligned}
&\mathsf{ti}(ack, t+d) = \langle sending\_data \rangle \ \wedge\ \mathsf{ti}(a, t+1) = \langle sc\_ack \rangle \\
&\wedge\ (\forall\, k \in \mathbb{N} :\ k \leq d+1 \rightarrow \mathsf{ti}(lose, t+k) = \langle \mathsf{false} \rangle) \\
&\qquad \rightarrow \mathsf{ti}(vc, t+d+1) = \langle vc\_com \rangle
\end{aligned}$$

4.5 Gateway: Architecture Specification

The specification of the gateway architecture, *Gateway*, is parameterized one: the parameter $d \in \mathbb{N}$ denotes the communication delay (between the central station and a vehicle). This component consists of three subcomponents: *Sample*, *Delay*, and *Loss*.

[12] The Focus operator $\mathsf{last}^{ti}(s,t)$ returns the last nonempty time interval of the stream s until the tth time interval. If until the time t all intervals were empty, the empty message list is returned.



We omit here the Focus specifications of this subcomponents and their translation in Isabelle/HOL, and give just a short description of them.

The component *Sample* represents the logic of the gateway component. If it receives from a ECall application of a vehicle the command to initiate the call to the Emergency Service Center it tries to establish the connection. If the connection is established, and the component *Sample* receives from a ECall application of a vehicle the command to send the crash data, which were already received and stored in the internal buffer of the gateway, these data will be resent to the Emergency Service Center. After that this component waits to the acknowledgment from the Emergency Service Center. If the acknowledgment is received, the voice communication will be established, assuming that there is no connection fails.

The component *Delay* models the communication delay. Its specification is parameterized one: it inherits the parameter of the component *Gateway*. This component simply delays all input messages on d time intervals. During the first d time intervals no output message will be produced.

The component *Loss* models the communication loss between the central station and the vehicle gateway: if during time interval t from the component *Loss_Oracle* no message about a lost connection comes, $\mathsf{ti}(lose, t) = \langle \mathsf{false} \rangle$, the messages come during time interval t via the input channels a and $i2$ will be forwarded without any delay via channels $a2$ and i respectively. Otherwise all messages come during time interval t will be lost.

4.6 Refinement Layers

The specification group *Automotive-Gateway* consists of the following components components: *GatewaySystemReq*, *GatewaySystem*, *ServiceCenter*, *GatewayReq*, *Gateway*, *Sample*, *Delay*, and *Loss*. The corresponding refinement layers of the specification group are presented on Figure 3.

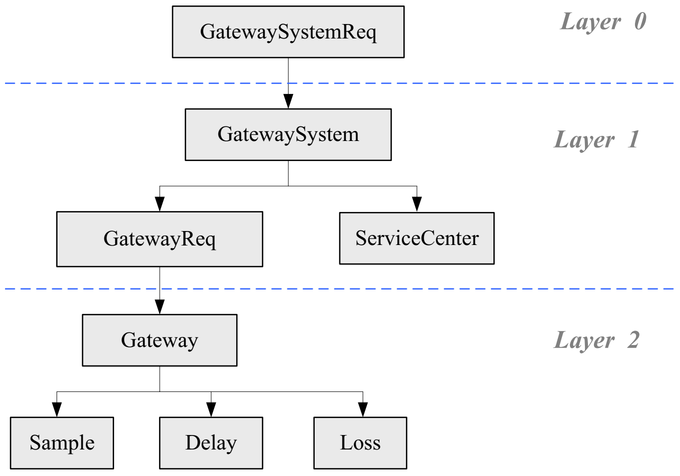


Fig. 3. Refinement Layers of the Specification Group *Automotive-Gateway*

4.7 Verification

In this section we discuss the proof that the specified automotive-gateway system architecture fulfills its specified requirements. First of all we need to show that the specified architecture of the gateway component itself fulfills the corresponding requirements. We define and prove the following lemma,[13] which says that the specification *Gateway* is a refinement of the specification *GatewayReq*:

lemma *Gateway_L0:*
Gateway req dt a stop lose d ack i vc $\Longrightarrow$
GatewayReq req dt a stop lose d ack i vc

To show that the specified system architecture fulfills the requirements we need to show that the specification *GatewaySystem* is a refinement of the specification *GatewaySystemReq*. Therefore, we define and prove the following lemma:[14]

lemma *GatewaySystem_L0:*
GatewaySystem req dt stop lose d ack vc $\Longrightarrow$

[13] The optimized proof is ca. 1200 lines of proof.
[14] The optimized proof is ca. 600 lines of proof.

$GatewaySystemReq\ req\ dt\ stop\ lose\ d\ ack\ vc$

In the proof of this lemma we have used first of all the definitions of the predicates *GatewaySystemReq* and *GatewaySystem*, and clarify the resulting goal. After that we add two new assumptions to the goal:

- The stream a has at very time interval at most one message.[15] This assumption is necessary as one of the gateway assumptions about the environment.
- The predicate *GatewayReq* holds for the corresponding streams, i.e. that the gateway fulfills its requirements (according lemma *Gateway_L0*). This assumption is needed to simplify the proof – now we can prove a number of system properties directly from the *properties* of the gateway, without extraction the definitions of the gateway architecture and the properties of its components.

Proving the lemma *GatewaySystem_L0*, we found out a number of gateway properties which can be seen as requirements to the gateway:

- If at the tth point in time the gateway has establish the connection, and it does not get any command to send the E-Call data to the central station until the $(t+k)$th time interval, and also there is no environment connection problems during these time intervals, then it stays it the same state waiting for the command to send the E-Call data.
- If at tth time interval the gateway is in the initial state, and at time interval $t+1$ the signal about crash comes at first time, and after $3+m$ time intervals the command to send the crash data comes at first time, and there is no connection fails from the time t until the $(t+3+k)$th time interval, then until the $(t+3+k+d)$th time interval the output stream i contains no messages.
- If before the tth point in time the gateway has send the E-Call data, but time interval became no acknowledgment from the central station until the $(t+d)$th point in time, and also there is no environment connection problems, then it stays it the same state waiting for the acknowledgment.

We can add these properties to the specification of the gateway requirements. The extended version of the gateway requirements specification *GatewayReq-Ext* is shown below (the new requirements are marked with green color).

The specified gateway architecture fulfills certainly the extended requirements, i.e. that the specification *Gateway* is a refinement of the specification by the

[15] This stream goes from the central station to the gateway, see Section 4.2.

following *lemma* (the specification *GatewayReqExt* is then translated schematically into the Isabelle/HOL predicate *GatewayReqExt*):[16]

lemma *Gateway_Ext:*
Gateway req dt a stop lose d ack i vc $\Longrightarrow$
GatewayReqExt req dt a stop lose d ack i vc

The specified requirements of the gateway component (specification *GatewayReq*) are not strong enough to prove the system properties without extraction the definitions of the gateway architecture and the properties of its components, but using the extended version of the gateway requirements specification, we can make such kind of proofs. This means that we can shift a part of proof to the upper refinement layer to optimize the proof structure.[17]

Thus, we need either replace the specification *GatewayReq* on the refinement layer 1 (see Figure 3) by the specification *GatewayReqExt* or add a new refinement layer with the extended version of the specification of requirements (see Fugure 4) to make the whole proof structure more clear and reusable.

[16] The optimized proof is ca. 1500 lines of proof, but the proofs of the new auxiliary lemmas are very similar to the proofs of the corresponding parts of the lemma *GatewaySystem_L0*.
[17] The optimized proof of the corresponding lemma *GatewaySystem_L0ext* is ca. 300 lines of proof.

GatewayReqExt(const d $\in \mathbb{N}$) timed

in $req : \{init\_connect\};\ dt : ECall\_Info;\ a : \{sc\_ack\};$
$stop : stop\_vc;\ lose : Bool$

out $ack : GatewayStatus;\ i : ECall\_Info;\ vc : \{voice\_com\}$

univ $k \in \mathbb{N}$

asm $\mathsf{msg}_1(req) \wedge \mathsf{msg}_1(a) \wedge \mathsf{msg}_1(stop) \wedge \mathsf{ts}(lose)$

gar

$\forall\, t \in \mathbb{N}:$

$$\mathsf{ti}(ack, t) = \langle init\_state \rangle \wedge \mathsf{ti}(req, t+1) = \langle init \rangle$$
$$\wedge\ \mathsf{ti}(lose, t+1) = \langle \mathsf{false} \rangle \wedge \mathsf{ti}(lose, t+2) = \langle \mathsf{false} \rangle$$
$$\rightarrow \mathsf{ti}(ack, t+2) = \langle connection\_ok \rangle$$

$$\mathsf{ti}(ack, t) = \langle init\_state \rangle \wedge \mathsf{ti}(req, t+1) = \langle init \rangle$$
$$\wedge\ \mathsf{ti}(req, t+3+k) = \langle send \rangle$$
$$\wedge\ \forall\, t_1 \le t:\ \mathsf{ti}(req, t1) = \langle\rangle \wedge \forall\, m \le k+3:\ \mathsf{ti}(req, t+m) \ne \langle send \rangle$$
$$\wedge\ \forall\, j \le k+d+3:\ \mathsf{ti}(lose, t+j) = \langle \mathsf{false} \rangle$$
$$\rightarrow \forall\, t_2 \le t+3+k+d:\ \mathsf{ti}(i, t_2) = \langle\rangle$$

$$\mathsf{ti}(ack, t) = \langle connection\_ok \rangle \wedge \forall\, m \le k:\ \mathsf{ti}(req, t+m) \ne \langle send \rangle$$
$$\wedge\ \forall\, j \le k:\ \mathsf{ti}(lose, t+j) = \langle \mathsf{false} \rangle$$
$$\rightarrow \forall\, y \le k:\ \mathsf{ti}(ack, t+y) = \langle connection\_ok \rangle$$

$$\mathsf{ti}(ack, t) = \langle connection\_ok \rangle \wedge \mathsf{ti}(req, t+1) = \langle send \rangle$$
$$\wedge\ (\forall\, k \in \mathbb{N}:\ k \le d+1 \rightarrow \mathsf{ti}(lose, t+k) = \langle \mathsf{false} \rangle)$$
$$\rightarrow \mathsf{ti}(i, t+d+1) = \mathsf{last}^{ti}(dt, t) \wedge \mathsf{ti}(ack, t+1) = \langle sending\_data \rangle$$

$$\mathsf{ti}(ack, t) = \langle sending\_data \rangle \wedge \forall\, t_3 \le t+d : \mathsf{ti}(a, t_3) = \langle\rangle$$
$$\wedge\ \forall\, j \le d+d:\ \mathsf{ti}(lose, t+j) = \langle \mathsf{false} \rangle$$
$$\rightarrow \forall\, x \le d+d:\ \mathsf{ti}(ack, t+x) = \langle sending\_data \rangle$$

$$\mathsf{ti}(ack, t+d) = \langle sending\_data \rangle \wedge \mathsf{ti}(a, t+1) = \langle sc\_ack \rangle$$
$$\wedge\ (\forall\, k \in \mathbb{N}:\ k \le d+1 \rightarrow \mathsf{ti}(lose, t+k) = \langle \mathsf{false} \rangle)$$
$$\rightarrow \mathsf{ti}(vc, t+d+1) = \langle vc\_com \rangle$$

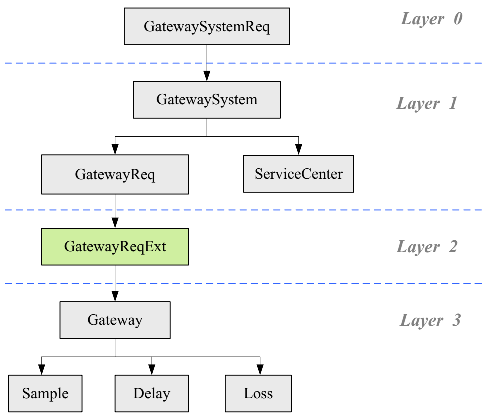


Fig. 4. Extended Refinement Layers of the Specification Group *Automotive-Gateway*

4.8 Results of the Case Study

In this case study we have shown how we can verify larger systems using the idea of the refinement-based verification. The Focus specifications of all components of the gateway system were translated schematically to Isabelle/HOL and the refinement relation between the requirement and the architecture specification was proved both for the gateway component and for the gateway system. We also present an example of extension of the requirements specification by the new properties found out during the verification. After that the proof of a number of system properties were done *directly from the properties* of the gateway, without extraction the definitions of the gateway architecture and the properties of its components. The proof of the gateway system properties using the extended version of the gateway requirements takes *ca. 50% of the proof for the non-extended version.*

5 Conclusion

In this paper we present the idea of the refinement-based verification. We treat any proof about a system as the proof that a more concrete system specification is a refinement of a more abstract one. The case when one needs to prove a single property of a system specification S can also be seen as a refinement relation.

We introduce the representation of the refinement layers of a specification group and the general ways of their representation in Isabelle/HOL as well as the definition of the strict behavioral refinement. After that we discuss the representation in Isabelle/HOL of different kinds of refinement – behavioral, interface, and conditional refinement – and how the ideas of the refinement-based verification can be used in the framework "Focus on Isabelle" [9]. Given two specifications, represented in a formal specification framework Focus, our method "Focus on Isabelle" validates the refinement relation between them by translating the specifications to a Higher Order Logic and subsequent using the theorem prover Isabelle/HOL.

The presented case study, verification of the gateway system specification, showed the feasibility of the approach. Doing the verification in Isabelle/HOL of the first versions of the Focus specification of the gateway system we found out a number of properties to extend the gateway requirement specification. The proof of the gateway system properties using the extended version of the gateway requirements takes *ca. 50% of the proof for the non-extended version.* Thus, using the idea of the refinement-based verification we get more clear and reusable proof structure.

References

[1] European Commission (DG Enterprise and DG Information Society): eSafety forum: Summary report 2003. Technical report, eSafety, 2003.

[2] J.-R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, New York, NY, USA, 1996.

[3] J. Botaschanjan, A. Gruler, A. Harhurin, L. Kof, M. Spichkova, and D. Trachtenherz. Towards Modularized Verification of Distributed Time-Triggered Systems. In *FM 2006: Formal Methods*, pages 163–178. Springer Verlag, 2006.

[4] M. Broy. Compositional refinement of interactive systems. *J. ACM*, 44(6), 1997.

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

[6] Manfred Broy. Compositional refinement of interactive systems modelled by relations. *COMPOS'97: Revised Lectures from the International Symposium on Compositionality: The Significant Difference*, 1998.

[7] C. Kühnel and M. Spichkova. Fault-Tolerant Communication for Distributed Embedded Systems. In *Software Engineering and Fault Tolerance*, Series on Software Engineering and Knowledge Engineering, 2007.

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

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

[10] Verisoft Project. http://www.verisoft.de.

[11] M. Wenzel. *The Isabelle/Isar Reference Manual.* Technische Universität München, 2004.