Focus on Isabelle: From Specification to Verification[⋆]

Maria Spichkova

Institut für Informatik, Technische Universität München
spichkov@in.tum.de

Abstract. This approach introduces a coupling of a specification framework with a verification system. Given a system, represented in a formal specification framework, 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 will be found. Moreover, using this approach one can validate the refinement relation between two given systems, as well as make automatic correctness proofs of syntactic interfaces for specified system components. The approach uses particularly the idea of refinement-based verification, where a verification of system properties can be treated as a validation of a system specification with respect to the specification of the properties.

1 Motivation

Embedded systems is one of the most challenging fields of systems engineering: such a system must most of the time meet real-time requirements, is safety critical and distributed. The current practice in the industry of ensuring that a software system fulfills its requirements is testing. However, testing can only demonstrate the presence, but not the absence of errors. Using formal methods we can not 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. Coupling a specification framework with a verification system will reduce the lavishness and error-proneness of system specifications. 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 – for this case *verified* formal specifications are needed. Having a verified formal specification we can be sure that the specification conforms to its requirements and is consistent.

In this paper we present a coupling of a specification framework with a verification system. Given system and requirements specifications, represented in a formal specification framework, 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. This approach contains also schemata for automatic correctness proofs in Isabelle/HOL of syntactic interfaces for specified system components.

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. A specification scheme of Focus is inspired by specification approaches like Z (see [25]), but the Focus framework is much more powerful and expressive – it supports a variety of specification styles which describe system components by logical formulas or by diagrams and tables representing logical formulas. Focus is preferred here over other specification frameworks since it has an integrated notion of time and modeling techniques for unbounded networks (specification replications, sheaves of channels), 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 approaches 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

[⋆] 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.

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 Isabelle/HOL we have chosen Isabelle/HOL [16,27], 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 methodology is presented in [23]. In this paper we show the application of its main contributions on the example of a verified specification of the FlexRay communication protocol. This protocol has been developed by the FlexRay Consortium for embedded systems in vehicles. The advantages of FlexRay over a CAN protocol (Controller Area Network), which is the most currently used protocol for such kind of systems, are deterministic real-time message transmission, fault tolerance, integrated functionality for clock synchronization and higher bandwidth. The Focus specification [11] of FlexRay was chosen for the case study because the protocol is very well suited for our method – its domain is safety-critical real-time applications.

Outline. The rest of the paper is structured as follows: In Section 2 we introduce Focus and the representation of its main concepts in Isabelle/HOL. In Section 3 we describe the application of represented ideas within a case study – a verified specification of the FlexRay. In Section 4 we present the related work. Finally, in Section 5 we summarize the presented work.

2 FOCUS on Isabelle

Isabelle [16] is a specification and verification system implemented in the functional programming language ML. Isabelle/HOL is the specialization of Isabelle for Higher Order Logic. To specify a system with Isabelle means creating *theories*. A theory is a named collection of types, functions (constants), and theorems (lemmas). 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, and `set`, the type of sets. Function types are denoted by $\Rightarrow$. 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. For a detailed description of Isabelle/HOL see [16,27].

Focus is a framework for formal specifications and development of distributed interactive systems. A distributed system in Focus is represented by its *components*[1]. 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. Messages are passed along the channels one after the other and delivered in exactly the same order in which they were sent.

In Focus any specification characterizes the relation between the *communication histories* for the external *input* and *output channels*. The formal meaning of a specification is exactly this external *input/output relation*. 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*. A specification can be elementary or composite. *Composite specifications* are built hierarchically from the elementary ones. *Elementary specifications* are divided into untimed, timed, and time-synchronous according to their level of time abstraction.

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,

[1] A component in Focus means a "logical component" and not a physical one.

whose results have helped us to find out different problem points (like representation of mutually recursive functions, specification replications, sheaves of channels, a large number of refinement layers, etc.) 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 the framework "Focus on Isabelle", which is presented here, 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.

2.1 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. Specifying embedded real-time systems we always need to argue about time. The notion of time takes center stage for this kind of systems and abstracting from time we may loose very important properties, e.g. the causality property, that are not only very important for the system, but also help us to make proofs easier. Thus, the timed domain is the most important one for representation of distributed systems with real-time requirements. Therefore, the better way to represent a real-time system in Focus is to use timed specification. All input, output and local streams in such specifications are timed or time-synchronous and, by the Focus definition of timed stream, infinite.

Specification of a real-time system in the untimed frame may be in some cases shorter or more elegant from mathematical point of view, but case studies have shown, that to understand such specifications and to argue about their properties is in many cases much more difficult in comparison to the corresponding specifications in the timed frame that use causality property explicitly. Moreover, abstraction from timing aspects can easily lead to specification mistakes because of difficulties of correct abstraction.

Hence, we can restrict the Focus specification domain for representation embedded real-time systems to only timed and time-synchronous systems. This simplifies the translation into Isabelle and also allows us to concentrate on the timing properties to have not only more clear and readable specifications, but also simpler proofs about them. Considering causality (weak or strong) it is simpler and also more readable to argue not about single messages in a timed stream, but about a sequence of messages that are present in this stream at some time interval. This sequence can be in general empty, contain a single message or a number of messages. In the case of time-synchronous stream this sequence must always contain exactly one message.

The definition in Isabelle/HOL of the Focus stream types is given below. Another ways of streams formalizations are discussed in Section 4.

- *Finite timed streams* of type `'a` are represented by the type `'a fstream`, which is an abbreviation for the type `'a list list`. This type will be used in Focus specifications of real-time systems to argue about a timed stream that was truncated at some point of time.
- *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.
- *Infinite timed streams* of type `'a` are represented by the type `'a istream` that represents the functional type `nat ⇒ 'a list`.
- To cover all types of Focus streams the type *infinite untimed streams* is also specified: `nat ⇒ 'a`. We do not advise to use this kind of streams to specify real-time systems, because a specification a real-time system to untimed domain implies the loss of a most important information about a system.

For easier argumentation about the behavior of a component at some time interval we have introduced a special kind of Focus tables and a number of new operators. Here only small part of them will be used, e.g. the operator $\mathsf{ti}(s, n)$ yields the list of messages that occur in the timed stream s at the nth time unit, the operator $\mathsf{msg}_n(s)$ holds for a timed stream s, if this stream contains at every time unit at most n messages. According to our representation of the timed FOCUS streams the operator $\mathsf{ti}(s, n)$ corresponds in Isabelle/HOL simply to *s n*, to represent the $\mathsf{msg}_n(s)$ operator we specify in Isabelle/HOL a predicate *maxmsg n s* that is equal to the FOCUS operator modulo syntax. The whole translation schema – from FOCUS to Isabelle/HOL – is presented in [23].

2.2 Sheaves of Channels

A specified system can contain a number of copies of channel of the same type or several instances of the same component. If this number of copies is finite, fixed and small enough, we can use the simple composition kinds, but if the number of copies must be specified as some variable of type $\mathbb{N}$ or if the number of copies is finite and fixed, but too large to have a readable system specification, the notions of *sheaf of channels* and *replication of specifications* must be used (see [5]). A sheaf of channels in FOCUS can be understood as an indexed set of channels.

We define a sheaf of channels $x_1, \dots, x_n$ as a *correct* one, if all the channels $x_1, \dots, x_n$ are of the same type and the number n is greater than zero. To represent a sheaf of timed infinite streams $x_1, \dots, x_n$ of some type *Streamtype* in Isabelle/HOL we propose to use the following kind of functional types:

```
types nStreamtype = "nat ⇒ streamtype istream"
```

The corresponding bounds of sheaves used to specify a component C will be added as extra-parameters to the Isabelle/HOL predicates which represent the semantics of the component C (see Sections 3.2 and 3.3 for examples).

A sheaf will be specified in Isabelle/HOL as a single variable of corresponding type, e.g. the sheaf $x_1, \dots, x_n$ will be represented as a variable *nX* of type *nStreamtype*. To translate the FOCUS formula over channels (streams) from a sheaf, e.g. to say that the predicate p is true for any stream of the sheaf $send_1, \dots, send_n$ (in FOCUS this formula is represented by $\forall\, i \in [1..n] :\ p(s_i)$) the following notation can be used[2]: *∀ i < n. p (nSend i)*.

To argue about sheaves of channels in Isabelle/HOL we need to make sure that the sheaf is nonempty[3]. For this propose the Isabelle/HOL predicate *CorrectSheaf n* is used. This predicate is true, if the number *n* of channels is greater than zero.

2.3 Specifications and the Concept of Refinement

FOCUS specifications can be *elementary* or *composite*. Syntax of an elementary specification looks like follows:

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 Section 2.1); *Parameter_Declarations* lists a number of parameters (optional); *Input_Declarations* and *Output_Declarations* list the declarations

[2] The relation < must be used, because the elements in Isabelle/HOL are counted from 0, in contrast to FOCUS, where the count goes from 1.

[3] In FOCUS this is automatically true: the notation $x_1, \dots, x_n$ implies that $0 < n$.

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 1. *For any timed elementary specification S we define its semantics, written $[\![S]\!]$, to be the formula:*

$$i_s \in I_S^{\infty} \ \wedge \ o_s \in O_S^{\infty} \ \wedge \ B_S \tag{1}$$

where i_S and o_S denote lists of input and output channel identifiers, I_S and O_S denote their corresponding types, and B_S is a formula in predicate logic that describes the Body. □

We define semantics of an elementary specification in Isabelle/HOL in the same way as it is defined in Focus: as a predicate that describes the relation between the input and output stream (the *Body*).

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

$$[\![S]\!] \overset{def}{=} \exists\, l_S \in L_S^{\infty} : \bigwedge_{j=1}^{n} [\![S_j]\!] \tag{2}$$

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

We define semantics of a composite specification in Isabelle/HOL analogous: a composite specification S is a predicate

$$\exists\, l_S \in istream_S : \bigwedge_{j=1}^{n} predicateS_j \tag{3}$$

where l_S denotes a list of *local channel identifiers* and $istream_S$ denotes their corresponding types, and $predicateS_j$ denotes the predicate is a representation in Isabelle/HOL of the Focus specification S_j.

The approach "Focus on Isabelle" contains a number of Isabelle/HOL theories and the corresponding schemata to prove correctness of the relations between the sets of input, output and local channels of a specified system. E.g., the following properties must be proven for every composite specification: No input stream i of a system S can be an output stream of any subcomponent.

$i_S = \bigcup_{j=1}^{n}(i_{S_j} \in I_S^{\infty}) \setminus l_S \ \wedge \ i_S \cap \bigcup_{j=1}^{n} o_{S_j} = \varnothing$

Every local stream l of the system S (consisting of n subcomponents) must be both an input stream of some subcomponent S_{j_1}, $1 \le j_1 \le n$, and an output stream of some subcomponent S_{j_2}, $1 \le j_2 \le n$ $(j_1 \neq j_2)$:

$l_S = \bigcup_{j=1}^{n} i_{S_j} \cap \bigcup_{j=1}^{n} o_{S_j}$

The proof schemata (for the correctness properties) specified in our approach are standard and can be used automatically for all refinement layers. If the proof fails, the specification of the corresponding set is incorrect and must be changed. But the main part of proofs about a system that we need are the proofs that a system fulfills its requirements.

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 (see also [5] and [3]) of the specification S_0. For this purposes also the idea of a refinement-based verification of interactive real-time systems can be used (see [23] and [24]).

Definition 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

$$[\![S_2]\!] \;\Rightarrow\; [\![S_1]\!] \tag{4}$$

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 $[\![S_2]\!]$ 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 and the refinement relation is true. This can happen if the specification S_2 is inconsistent. Thus, the consistency of S_2 must also be proved doing the proof in Isabelle/HOL.

3 Case Study: From Specification to Verification of the FlexRay Communication Protocol

The feasibility of the approach "Focus on Isabelle/HOL" was evaluated on a number of case studies that cover different application areas: process control, memory and processing components, data transmission etc. The proofs for these case studies have taken from 200 to ca. 2000 lines of proof. The following has been done within every case study: The Focus specifications of all components of the system have been translated schematically to Isabelle/HOL and the refinement relation between the requirement and the architecture specification of the system has been proved. The correctness of the input/output relations has been also proved for all components of the system (automatically, according to the specified proof schemata). The FlexRay case study was chosen for the case study to show how we can deal with sheaves of channels ans parameters, as well as with specification replications.

FlexRay [8] is a static time division multiplexing network protocol developed for embedded systems in vehicles. It is based on deterministic real-time message transmission between a number of nodes. FlexRay contains a set of complex algorithms to provide the communication services. From the view of the software layers above FlexRay only a few of these properties become visible. The most important ones are static cyclic communication schedules and system-wide synchronous clocks. These provide a suitable platform for distributed control algorithms as used e.g. in drive-by-wire applications. The formalization described here is based on the "Protocol Specification 2.0"[8]. A formal verification of the clock synchronization algorithm and of the bus guardian of FlexRay is in progress at INRIA [28].

The static message transmission model of FlexRay is based on *rounds*. FlexRay rounds consist of a constant number of time slices of the same length, so called *slots*. A node can broadcast its messages to other nodes at statically defined slots. At most one node can do it during any slot.

We have presented the first version of the formal specification of FlexRay in Focus in [11]. We have discussed the general introduction to the FlexRay formalization also in [12] and [13]. Now we are going to present a schematically translation of this formal specification into Isabelle/HOL using the representation of Focus streams presented above. After that the proof of the refinement lemma is discussed – the refinement lemma says that the FlexRay architecture specification fulfills the FlexRay requirements. Since the overall representation of FlexRay 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 [23].

3.1 Representation of Datatypes

The specifications of the following types are equal modulo syntax to the corresponding types in the Focus specification of FlexRay. The type *Message* consists of a slot identifier *slot* and the payload *data*. The type of payload is defined in Focus as a finite list of type *FT_CNI_Entity* that consists of a message ID of $\mathbb{N}$ (type of natural numbers) and data of type *DataType*. Because the type *DataType* is not specified in

FOCUS exactly (to have a polymorphic type), it must be polymorphic in Isabelle/HOL also. This implies furthermore that the types *FT_CNI_Entity* and *Message* are polymorphic. The type *Config* represents the bus configuration and contains the scheduling table `schedule` of a node and the length of the communication round `cycleLength`. A scheduling table of a node consists of a number of slots in which this node should be sending a message with the corresponding identifier (identifier that is equal to the slot). We present here as example only the Isabelle/HOL representation of the type *Message*:

```
record 'a Message =
   slot :: nat
   data :: "('a FT_CNI_Entity) list"
```

The types `nMessage` and `nSlot` are used to represent sheaves of channels of corresponding types. In a similar way we define the type `nConfig` for the list of parameter constants $c_1, \dots, c_n$ of the type *Config*.

```
  types 'a nMessage = "nat ⇒ ('a Message) istream"
  types nSlot = "nat ⇒ nat istream"
  types nConfig = "nat ⇒ Config"
```

3.2 Requirements Specification

The requirements specification *FlexRay*, which is represented in FOCUS, contains the assumptions (asm) and guarantees (gar) for the FlexRay network (an A/G specification). This means whenever input from the environment behaves in accordance with the assumption, the specified component is required to fulfill the guarantee. The assumptions are the following ones:

- For all nodes the scheduling tables of the system are disjoint.
- The communication cycles have the same length on each node.
- In every time interval on each input channel $return_i$ can come at most one FlexRay frame (a message of type *Message*).

Having this assumptions the specification requires fulfillment of the following properties:

- The message transmission is correct: If at some slot a node should be sending a message according to it's scheduling table, this message is requested from the local buffer and is sent over the channels to the other nodes of the system.
- In every time interval on each output channel get_i as well as can come on each output channel $store_i$ at most one FlexRay frame.

FlexRay (const $\mathrm{c}_1, ..., \mathrm{c}_\mathrm{n} \in$ Config) timed

in $return_1, ..., return_n : Message$

out $store_1, ..., store_n : Message;\ get_1, ..., get_n : Slot$

asm $\forall\, i \in [1..n] : \mathsf{msg}_1(return_i)$
$DisjointSchedules(c_1, ..., c_n)$
$IdenticCycleLength(c_1, ..., c_n)$

gar
$MessageTransmission(return_1, ..., return_n, store_1, ..., store_n, get_1, ..., get_n, c_1 ..., c_n)$
$\forall\, i \in [1..n] : \mathsf{msg}_1(get_i) \wedge \mathsf{msg}_1(store_i)$

The FOCUS predicate *MessageTransmission* defines the correct message transmission according the definition above: If at time t the node k should be sending a message according to its scheduling table, this message is requested over the channel get_k and received over the channel $return_k$ from the local buffer. This message is then sent over the channels $store_j$, $j \in [1..n]$, $j \neq k$ to the other nodes of the system.

The predicates *DisjointSchedules*, *IdenticCycleLength*, *MessageTransmission* from the FOCUS specifications are equal modulo syntax to predicates the same name that we specify in Isabelle/HOL. The predicate *DisjointSchedules* is defined to be true for a sheaf of channels, if all bus configurations have disjoint scheduling tables. The predicate *IdenticCycleLength* is defined to be true, if all bus configurations have the equal length of the communication round.

The predicate `FlexRay` represents the semantics of the FOCUS specification *FlexRay*. In the case the relation between the input and output streams is specified in FOCUS as a number of formulas the corresponding representation of the semantics will be the conjunction of these formulas. The first argument of the predicate `FlexRay` corresponds to the number n of streams in the sheaves $store_1, ..., store_n$, $get_1, ..., get_n$, $return_1, ..., return_n$ and the number of parameters $c_1, ..., c_n$.

constdefs

```
  FlexRay ::
    "nat ⇒ 'a nMessage ⇒ nConfig ⇒ 'a nMessage ⇒ nSlot ⇒ bool"
 "FlexRay n nReturn nC nStore nGet
  ≡
   (CorrectSheaf n  ∧
   (∀ i < n. maxmsg 1 (nReturn i)) ∧
   (DisjointSchedules n nC) ∧ (IdenticCycleLength n nC)
    ⟶
   ((MessageTransmission n nReturn nStore nGet nC)  ∧
   (∀ i < n. maxmsg 1 (nGet i) ∧ maxmsg 1 (nStore i))))"
```

3.3 Architecture Specification

The architecture of the FlexRay communication protocol is specified as the the FOCUS specification *FlexRayArch* that is an assumption/guarantee one. The assumption part of the specification *FlexRayArch* is the same as of the specification *FlexRay*. The guarantee part is represented by the specification *FlexRayArchitecture* (see below) that is a composite one and consists of the component *Cable* and n components *FlexRay_Controller* (for n nodes). The specification *FlexRayArch* is a refinement of the specification *FlexRay* – this will be shown in Section 3.4.

FlexRayArch (const $c_1, ..., c_n \in$ Config) — timed

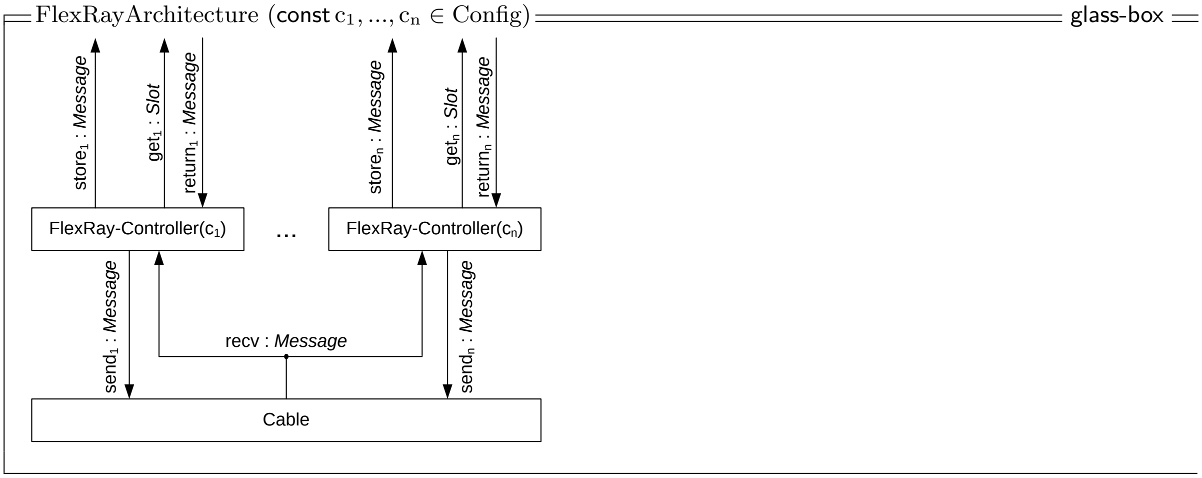
in $return_1, ..., return_n : Message$

out $store_1, ..., store_n : Message$; $get_1, ..., get_n : Slot$

asm $\forall i \in [1..n] : \mathsf{msg}_1(return_i)$
$DisjointSchedules(c_1, ..., c_n)$
$IdenticCycleLength(c_1, ..., c_n)$

gar *FlexRayArchitecture* (const $c_1, ..., c_n \in Config$)
$(return_1, ..., return_n, store_1, ..., store_n, get_1, ..., get_n)$

The component *FlexRay-Controller* is a composite one and consists of the components *Scheduler* (has a timing control function: decides if the node is allowed to send in the current slot) and *BusInterface* (represents the receive and the send of messages). The component *Cable* describes the transfer properties: in every time unit only one of the streams has any messages to transfer, if some node sends a message, this message will be received by all nodes, etc. The predicate `FlexRayArchitecture` represents here the semantics of the corresponding FOCUS specification. Please note, that *nSend* and *recv* are the local channels in the composite component.



constdefs

```
  FlexRayArchitecture ::
    "nat ⇒ 'a nMessage ⇒ nConfig ⇒ 'a nMessage ⇒ nSlot ⇒ bool"
 "FlexRayArchitecture n nReturn nC nStore nGet
 ≡
 ∃ nSend recv.
   CorrectSheaf n ∧ (Cable n nSend recv) ∧
   (∀ i < n. FlexRay_Controller (nReturn i) recv (nC i)
            (nStore i) (nGet i) (nSend i))"
```

3.4 Proof of the Refinement in Isabelle/HOL

The lemma `main_fr_refinement` says that the specification *FlexRayArch* is refinement of the specification *FlexRay*: the predicate `FlexRayArch` that represents the semantics of the architecture specification *FlexRayArch* implies the `FlexRay` that represents the semantics of the requirements specification *FlexRay*:

lemma `main_fr_refinement:`

```
⋀ n nReturn nC nStore nGet.
FlexRayArch n nReturn nC nStore nGet ⟹ FlexRay n nReturn nC nStore nGet
```

To prove this lemma we used the definitions of the predicates, used in the specification, Isabelle/HOL reasoning methods `clarify`, `clarsimp` and `auto` that perform rewriting and classical reasoning automatically, rules of natural deduction etc. To prove the resulting subgoals in more structural way, we have proven a number of auxiliary lemmas [23]. The proof of this lemma takes only ca. 300 lines, but rather in a specification of such an observable size we have found out a number of inconsistencies (see the next subsection).

3.5 Results of the Case Study

In this case study we have shown how we can deal with sheaves of channels and parameters, as well as with specification replications. The FOCUS specifications of all components of the FlexRay system were translated schematically to Isabelle/HOL and the refinement relation between the requirement and the architecture specification of the system was proved. The correctness of the input/output relations was also proved for all components of the system.

Doing the verification in Isabelle/HOL of the first versions of the Focus specification of FlexRay we found out a number of inconsistencies, which were corrected in the final version: lost assumptions, too weak properties of the subcomponents, etc. For example, in the requirement specification *FlexRay* the assumption, that in every time interval on each input channel $return_i$ can come at most one FlexRay frame, was loosed. The resulting formal specification of FlexRay is a *verified* specification that guarantees fulfillment of the requirements.

4 Related Work

The first attempt to represent a simplified version of the Focus syntax [4] (without representation of time, modeling techniques for unbounded networks, etc.) in a verification system was done by B. Schätz and K. Spies [20]. In this approach the HOLCF specialization of the theorem prover Isabelle was chosen. HOLCF (see [19] and [15]) is the definitional extension of Church's Higher-Order Logic with Scott's Logic for Computable Functions that has been implemented in Isabelle. HOLCF supports standard domain theory but also coinductive arguments about lazy datatypes. The main disadvantage of using HOLCF in practice is difficulty of logic understanding in comparison to HOL.

The first attempt of coupling of Focus with an *automatic* verification system was done by J. Schumann and M. Breitling [21]. As the verification system was chosen SETHEO [14], an automatic theorem prover for proving the unsatisfiability of formulas in First-Order Clause Logic. This case study of J. Schumann and M. Breitling has shown that such a coupling is in principle possible, but there are also a number of problems and open questions. 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.

The central concept in Focus are streams, and there are different ways to formalize them. They have different advantages and disadvantages. One of the ways to represent Focus streams is to use the coalgebraic approach [10], but the representation of Focus streams in a coalgebraic domain [22] is more difficult to understand in practice as an inductive one in Isabelle/HOL – for the case of restriction the specification domain to only real-time systems.

The representation of Focus streams in Isabelle/HOLCF that was done by D. von Oheimb [26] does not cover representation of Focus timed streams, which are the most important for the specification of real-time systems. The further development of the Focus stream representation in Isabelle/HOLCF is presented by the approach of B. Gajanovic and B. Rumpe [9], that covers HOLCF specification of many important operators on streams, like concatenation, delete prefixes, take an element of the stream etc., as well as the properties of these operators. But this representation of the Focus streams in Isabelle/HOLCF covers only the general representation of streams, and abstracts from the representation of timing aspects as well as from the question how to deal with proofs for such translated specifications. Thus, the representation of the timing aspects can be done as an extension, but the resulting construction will be much more complicated than is needed for system specification in the timed domain.

To represent Focus streams in Isabelle/HOL we need to take into account both properties of Focus and Isabelle/HOL. In Isabelle/HOL we can represent streams in the two following ways. The first way is the representation of streams as

$$\alpha\ \texttt{seq} = \mathbb{N} \to \alpha\ \texttt{option}$$

where the datatype

$$\alpha\ \texttt{option} \equiv \texttt{None} \mid \texttt{Some}\ \alpha$$

and `None` denotes a non-existing element (see [18], [17]). This approach is claimed in [7] to be inconvenient in practice to prove equalities of arbitrary functions, because for every operation over such a stream the notion of stream normal form must be used explicitly to avoid the case in which `None` appears within a sequence the specification, but it is not straightforward to construct the normal form.

The second way, the representation of streams as the disjoint sum of finite stream (lists) and infinite streams (functions), has been chosen by C.-T. Chou and D. Peled [6] and by S. Agerholm [2]. The main difficulties (see also [7]) in using this approach arise from type comparison in the definitions of stream processing functions – all inputs are finite, all inputs are infinite or some of them are finite and some infinite – several versions of function definitions are needed. But in the case we work only with timed frames we do not have this disadvantage, because we deal with timed streams that are always infinite[4]. Moreover, such a representation in this case leads to more clear specification structure. This representation is the most natural one to Focus and was in our approach for the our representation of Focus in Isabelle/HOL.

5 Conclusion

A formal specification is 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 – in this case formal verification is needed. This is the only way to be sure that the specification conforms to its requirements and is consistent. In this paper we have introduced the coupling of the formal specification framework Focus in the generic theorem prover Isabelle/HOL. The result of the coupling of the formal specification framework Focus in the generic theorem prover Isabelle/HOL is the framework "Focus on Isabelle". Given both specifications represented in Focus, we prove using the theorem prover Isabelle/HOL that the system specification is a refinement of requirements specification of the system, i.e. that this specification fulfills its requirements. Using the framework we also can make automatic correctness proofs in Isabelle/HOL of the syntactic interfaces for specified system components.

The presented case study, verification of the FlexRay specification, showed the feasibility of the approach. The properties of the protocol were formalized as the requirements specification, and its architecture was formalized as the corresponding specification. The system specification was subsequently verified according to the FlexRay requirements – the refinement relation between them is proved. Doing the verification in Isabelle/HOL of the first versions of the Focus specification of FlexRay we found out a number of inconsistencies like loosed assumptions, which were corrected in the final version of the specification. The resulting specification is a *verified* one.

References

1. J.-R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, New York, NY, USA, 1996.
2. S. Agerholm. *A HOL Basis for Reasoning about Functional Programs.* PhD thesis, University of Aarhus, 1994.
3. M. Broy. Compositional refinement of interactive systems. *J. ACM*, 44(6):850–891, 1997.
4. M. Broy, F. Dederich, C. Dendorfer, M. Fuchs, T. Gritzner, and R. Weber. The Design of Distributed Systems – An Introduction to FOCUS. Technical Report TUM-I9202, Technische Univerität München, 1992.
5. M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001.
6. Ching-Tsun Chou and D. Peled. Formal verification of a partial-order reduction technique for model checking. In *TACAS*, pages 241–257, 1996.
7. M. Devillers, D. Griffioen, and O. Müller. Possibly Infinite Sequences in Theorem Provers: A Comparative Study. In Elsa Gunther, editor, *Theorem Proving in Higher Order Logics (TPHOL'97)*, number LNCS 1275. Springer, 1997.
8. FlexRay Consortium. *FlexRay Communication System – Protocol Specification – Version 2.0*, 2004.
9. B. Gajanovic and B. Rumpe. Isabelle/HOL-Umsetzung strombasierter Definitionen zur Verifikation von verteilten, asynchron kommunizierenden Systemen. Technical Report 2006-03, Technische Universität Braunschweig, 2006.

[4] The timed streams must be infinite because time never halts.

10. B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62:222–259, 1997.
11. C. Kühnel and M. Spichkova. FlexRay und FTCom: Formale Spezifikation in Focus. Technical Report TUM-I0601, Technische Universität München, 2006.
12. C. Kühnel and M. Spichkova. Upcoming Automotive Standards for Fault-Tolerant Communication: FlexRay and OSEKtime FTCom. In *EFTS 2006 International Workshop on Engineering of Fault Tolerant Systems*. Universite du Luxembourg, CSC: Computer Science and Communication, 2006.
13. 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.
14. R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: a high-performance theorem prover. *J. Autom. Reason.*, 8(2):183–212, 1992.
15. O. Müller, T. Nipkow, D. von Oheimb, and O. Slotosch. HOLCF = HOL + LCF. *Journal of Functional Programming*, (9(2)):191 – 223, 1999.
16. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
17. T. Nipkow and K. Slind. I/O Automata in Isabelle/HOL. In P. Dybjer, editor, *Proc. of Types for Proofs and Programs*, number LNCS 996, 1995.
18. L.C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer, 1994.
19. F. Regensburger. *HOLCF: eine konservative Erweiterung von HOL um LCF*. PhD thesis, Technische Universität München, 1994.
20. B. Schätz and K. Spies. Formale Syntax zur logischen Kernsprache der FOCUS-Entwicklungsmethodik. Technical Report TUM-I9529, Technische Universität München, 1995.
21. J. Schumann and M. Breitling. Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie. Technical Report TUM-I9904, Technische Universität München, 1999.
22. M. Spichkova. A coalgebraic view at data flow systems. Master's thesis, Technische Universität Dresden, 2003.
23. M. Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle*. PhD thesis, Technische Universität München, 2007.
24. M. Spichkova. Refinement-based verification of interactive real-time systems. In *REFINE 2008 – International Refinement Workshop*. ENTCS, 2008.
25. M. Spivey. Understanding Z – A Specification Language and Its Formal Semantics. Cambridge Tracts in Theoretical Comp. Science 3. Cambridge Univ. Press, 1988.
26. D. von Oheimb. Isabelle/HOLCF Formalization of Focus Streams. `http://isabelle.in.tum.de/library/HOLCF/FOCUS/Fstream.html`, 2005.
27. M. Wenzel. *The Isabelle/Isar Reference Manual*. Technische Universität München, 2004.
28. Bo Zhang. On the Formal Verification of the FlexRay Communication Protocol. In *Automatic Verification of Critical Systems (AVoCS)*, pages 184–189, 2006.