An Exercise in Conditional Refinement

Ketil Stølen[1] and Max Fuchs[2]

[1] OECD Halden Reactor Project, Institute for Energy Technology, P.O.Box 173, N-1751, Halden, Norway
[2] BMW AG, FIZ EG-K-3, D-80788, München, Germany

Abstract. This paper is an attempt to demonstrate the potential of conditional refinement in step-wise system development. In particular, we emphasise the ease with which conditional refinement allows boundedness constraints to be introduced in a specification based on unbounded resources. For example, a specification based on purely asynchronous communication can be conditionally refined into a specification using time-synchronous communication.
The presentation is built around a small case-study: A step-wise design of a timed FIFO queue that is partly to be implemented in hardware and partly to be implemented in software. We first specify the external behaviour of the queue ignoring timing and synchronisation. This overall specification is then restated in a time-synchronous setting and thereafter refined into a composite specification consisting of three subspecifications: A specification of a time-synchronous hardware queue, a specification of an asynchronous software queue, and a specification of an interface component managing the communication between the first two. We argue that the three overall specifications can be related by conditional refinement. By further steps of conditional refinement additional boundedness constraints are introduced. We explain how each step of conditional refinement can be formally verified in a compositional manner.

1 Introduction

Requirements imposing upper bounds on the memory available for some data structure, channel, or component are often programming language or platform dependent. Such requirements may for instance characterise the maximum number of messages that can be sent along a channel within a time unit without risking malfunction because of buffer overflow. Clearly, this number may vary from one buffer to another depending on the type of messages it stores, and the way the buffer is implemented.

One way to treat such boundedness constraints is to introduce them already at the most abstract level — in other words, in the requirement specification. However, this option is not very satisfactory, for several reasons. The boundedness constraints:

- considerably complicate the specifications; as a result, the initial understanding of the system to be designed is often reduced;

B. Möller and J.V. Tucker (Eds.): Prospects for Hardware Foundations, LNCS 1546, pp. 390-420, 1998.
© Springer-Verlag Berlin Heidelberg 1998

- reduce the set of possible implementations; this makes the reuse of specifications more difficult;
- complicate formal reasoning; this means it becomes more difficult to verify the correctness of refinement steps;
- are often not known when the requirement specifications are written; for instance, at this stage in a system development, it is often not decided which implementation language(s) to use and on what sort of platform the system is to run.

We conclude that any development method requiring that boundedness constraints are introduced already in the requirement specification, is not very useful from a practical point of view. However, since any computerised system is based on bounded resources, we need concepts of refinement supporting the transition from specifications based on unbounded resources to specifications based on bounded resources. This paper advocates the flexibility of conditional refinement for this purpose. We define conditional refinement for a specification language based on streams. A specification in this language is basically a relation between input and output streams. Each stream represents a complete communication history for a channel. To formalise timing properties we use what we call timed streams. Timed streams capture a discrete notion of time. The underlying communication paradigm is asynchronous message passing. The ideas of this paper can be adapted to other specification languages, as well. For example, [14] formulates a notion of conditional refinement in SDL (see also [7]).

The remainder of the paper is organised as follows: In Sect. 2 we introduce the basic concepts and notations; in Sect. 3 we give three different specifications of a FIFO queue; in Sect. 4 we formally define conditional refinement and explain how the specifications from Sect. 3 can be related by this concept; in Sect. 5 we conditionally refine the FIFO queue specified in Sect. 3 into a more concrete specification with additional boundedness constraints; in Sect. 6 we give a brief summary and relate our approach to other approaches known from the literature; in App. A we formulate three rules that can be used to verify the refinement steps mentioned above; in App. B we prove that these three rules are sound.

2 Basic Concepts and Notations

Streams model the communication histories of channels. We use both *timed* and *untimed streams*. A timed stream is a finite or infinite sequence of messages and *time ticks*. A time tick is denoted by $\surd$. The time interval between two consecutive ticks represents the least unit of time. A tick occurs in a stream at the end of each time unit.

An infinite timed stream represents a *complete communication history*, a finite timed stream represents a *partial communication history*. Since time never halts, we require that any infinite timed stream has infinitely many ticks.

By $M^{\underline{\infty}}$, $M^{\underline{*}}$ and $M^{\underline{\omega}}$ we denote respectively the set of all infinite timed streams, the set of all finite timed streams, and the set of all finite and infinite timed streams over the set of messages M.

We also work with streams without ticks, referred to as untimed streams. By M^∞, M^* and M^ω we denote respectively the set of all infinite untimed streams, the set of all finite untimed streams, and the set of all finite and infinite untimed streams over the set of messages M. In the sequel, M denotes the set of all messages. Since $\surd$ is not a message, we have that $\surd \notin M$.

By $\mathbb{N}$ we denote the set of natural numbers; by $\mathbb{N}_\infty$ and $\mathbb{N}_+$ we denote $\mathbb{N} \cup \{\infty\}$ and $\mathbb{N} \setminus \{0\}$, respectively. Given $A \subseteq M \cup \{\surd\}$, streams $r, s \in M^\omega \cup M^{\underline{\omega}}$, $t \in M^{\underline{\infty}}$ and $j \in \mathbb{N}_\infty$:

- $\langle\rangle$ denotes the empty stream.
- $\#r$ denotes the length of r: ∞ if r is infinite, and the number of elements in r otherwise. Note that time ticks are counted. For example, the length of a stream consisting of infinitely many ticks is ∞.
- $r.j$ denotes the jth element of r if $1 \leq j \leq \#r$.
- $\langle a_1, a_2, .., a_n \rangle$ denotes the stream of length n, whose first element is a_1, whose second element is a_2, and so on.
- $A \circledS r$ denotes the result of filtering away all messages (ticks included) not in A. For example:

$$\{a, b\} \circledS \langle a, b, \surd, c, \surd, a, \surd \rangle = \langle a, b, a \rangle$$

- $r \frown s$ denotes the result of concatenating r and s. For example: $\langle c, c \rangle \frown \langle a, b \rangle = \langle c, c, a, b \rangle$. If r is infinite then $r \frown s = r$.
- r^j denotes the result of concatenating j copies of the stream r.
- $r \sqsubseteq s$ holds iff r is a prefix of or equal to s.
- $t{\downarrow_j}$ denotes the prefix of t characterising the behaviour until time j; this means that $t{\downarrow_j}$ denotes t if j is greater than the number of ticks in t, and the shortest prefix of t containing j ticks, otherwise. Note that $t{\downarrow_\infty} = t$, and also that $t{\downarrow_0} = \langle\rangle$.
- $\overline{t}$ denotes the result of removing all ticks in t. Thus, $\overline{\langle a, \surd, b, \surd \rangle \frown \langle\surd\rangle^\infty} = \langle a, b \rangle$.

A timed infinite stream is *time-synchronous* if exactly one message occurs in each time interval. Time-synchronous streams model the pulsed communication typical for synchronous digital hardware working in fundamental mode. Since, in this case, exactly one message occurs in each time interval, no information is gained from the ticks; the ticks can therefore be abstracted away; the result is an untimed infinite stream. To facilitate time abstraction in the time-synchronous case, the $\downarrow$ operator is overloaded to untimed infinite streams: For any stream $s \in M^\infty$ and $j \in \mathbb{N}_\infty$, $s{\downarrow_j}$ denotes the prefix of s of length j.

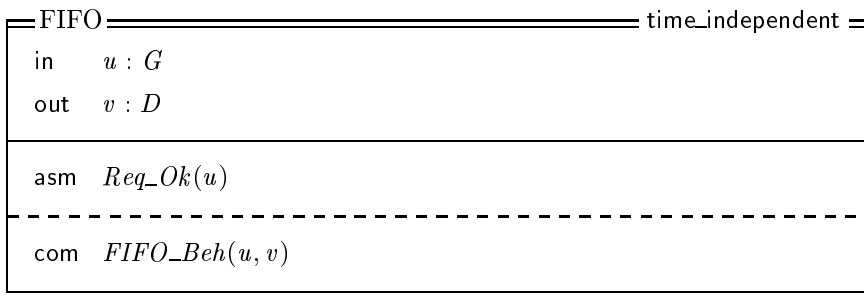
3 Specification of a FIFO Queue

Elementary specifications are basically relations on streams. These relations are expressed by formulas in predicate calculus. This section introduces three formats for elementary specifications: The *time-independent*, the *time-dependent*

and the *time-synchronous format*. We first specify the external behaviour of a FIFO queue in the time-independent format, a format tuned towards asynchronous communication. This specification is thereafter reformulated in the time-synchronous format; this format is particularly suited for the specification of digital hardware components communicating in a synchronous (pulsed) manner. Thereafter, we give a composite specification of the FIFO queue consisting of three elementary specifications: A *hardware queue* written in the time-synchronous format, a *software queue* written in the time-independent format, and an *interface component* written in the time-dependent format; the latter manages the communication between the first two.

3.1 Time-Independent Specification

The elementary specification below describes a FIFO queue.



FIFO is the name of the specification. Ignoring the dashed line, the specification FIFO is divided in two main parts by a single horizontal line. The upper part declares the *input* and *output channels* distinguished by the keywords in and out. Hence, FIFO has one input channel u of type G and one output channel v of type D. D is the set of all data elements. What exactly these data elements are is of no importance for this paper and therefore left unspecified. A request is represented by Req. It is assumed that $Req \notin D$. G is defined as follows:

$$G \equiv D \cup \{Req\}$$

Thus, data elements and request are received on the input channel; the queue replies by sending data elements. We often refer to the identifiers naming the channels as *channel identifiers*.

The lower frame, referred to as the *body*, describes the allowed input/output history. It consists of two parts: An *assumption* and a *commitment* identified by the keywords asm and com, respectively. The assumption describes the intended input histories — it is a pre-condition on the input histories. The commitment describes the output histories the queue is allowed to produce when the input history fulfills the assumption. Both the assumption and the commitment are described by formulas in predicate calculus. In these formulas u and v are free variables of type G^ω and D^ω, respectively; they represent the communication

histories of the channels they name. That u and v represent untimed streams (and, for instance, not timed infinite streams, as in the time-dependent case) is specified by the keyword time_independent in the upper right-most corner.

Assumption The assumption is expressed by an auxiliary predicate

$Req\_Ok(a)$

This predicate states that for any prefix b of the untimed stream a, the number of requests in b is less than or equal to the number of data elements in b. Formally:

Req_Ok

$$a \in M^{\omega} \cup M^{\underline{\infty}}$$

$$\forall\, b \in M^{\omega} \cup M^{\underline{\omega}} : b \sqsubseteq a \Rightarrow \#(\{Req\} \circledS b) \leq \#(D \circledS b)$$

Thus, the assumption of FIFO formalises that, at any point in time, the number of received requests is less than or equal to the number of received data elements; in other words, the environment is assumed never to send a request when the queue is empty. Note that a is of type $M^{\omega} \cup M^{\underline{\infty}}$ and not just G^{ω} $(\subseteq M^{\omega})$. This supports the reuse of *Req_Ok* in other contexts. Hence, the definition of the auxiliary predicate *Req_Ok* is independent from the specification FIFO in the sense that *Req_Ok* can be exploited also in other specifications.

Commitment Also the commitment employs an auxiliary predicate

$FIFO\_Beh(a, b)$

This predicate requires that the stream of data-elements sent along b is a prefix of the stream of data elements sent along a (first conjunct), and that the number of data elements in b is equal to the number of requests in a (second conjunct). Formally:

FIFO_Beh

$$a, b \in M^{\omega} \cup M^{\underline{\infty}}$$

$$D \circledS b \sqsubseteq (D \circledS a) \wedge \#D \circledS b = \#(\{Req\} \circledS a)$$

Thus, the commitment of FIFO requires that the data elements are output in the FIFO order, and that each request is replied to by the transmission of exactly one data-element.

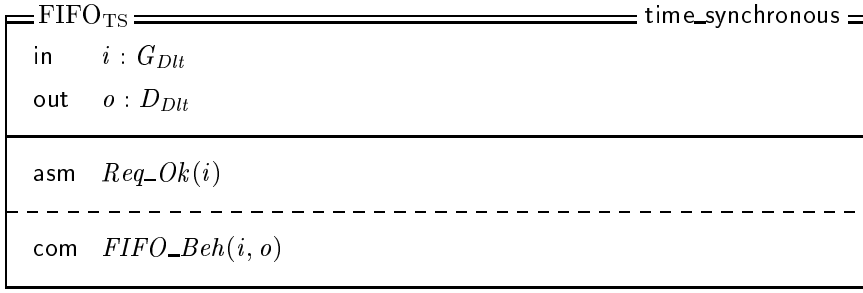
The semantics of elementary specifications is defined formally in Sect. 4.2.

3.2 Time-Synchronous Specification

The time-independent format employed in the previous section is well-suited to specify software components communicating asynchronously. To describe hardware applications communicating in a time-synchronous (pulsed) manner, we use the time-synchronous format as in this section. We now restate FIFO in a time-synchronous setting. Time-synchronous communication is modelled as follows: In each time unit exactly one message is transmitted along each channel. For the case that the time-synchronous FIFO queue or its environment have no "real" message to transmit, a default message Dlt is used. It is assumed that $Dlt \notin G$. We define:

$$G_{Dlt} \equiv G \cup \{Dlt\}, \qquad D_{Dlt} \equiv D \cup \{Dlt\}$$

FIFO can then be restated in the time-synchronous format as follows.



The keyword in the upper right corner implies that the channel identifiers in the body represent infinite untimed streams instead of untimed streams of arbitrary length as in FIFO. The filtration with respect to D in the definitions of the two auxiliary predicates of the previous section makes their reuse in FIFO_{TS} possible.

3.3 Composite Specification

We have presented elementary specifications written in the time-independent and the time-synchronous format. As already mentioned, there is also a time-dependent format. The time-dependent format is tuned towards asynchronous communication. It differs from the time-independent format in that it allows timing requirements to be expressed. We now use all three formats for elementary specifications to describe a composite FIFO queue that is partly to be implemented in hardware and partly to be implemented in software.

As illustrated by Fig. 1, the queue consists of three components, namely a hardware queue specified by HWQ, a software queue specified by SWQ, and an interface component specified by INTF. The interface component is needed as a converter between the time-synchronous hardware component and the asynchronous software component.

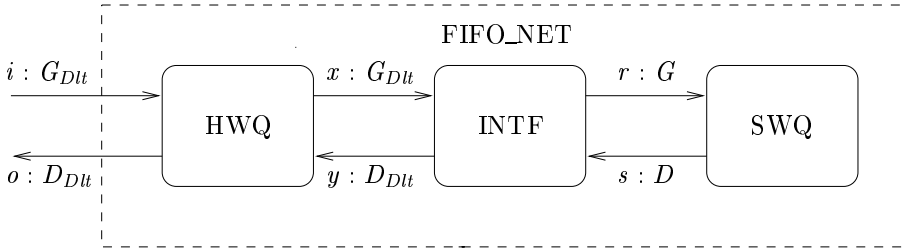


Fig. 1. Decomposed FIFO Queue

The hardware queue has only a bounded amount of internal memory: It can never store more than W_h data elements at the same point in time. To avoid memory overflow it may forward data elements and requests to the software queue. The software queue is in principle unbounded. The external behaviour of INTF and SWQ composed should be that of FIFO_{TS}.

Hardware Queue The hardware queue communicates in a time-synchronous manner: In each time-unit the hardware queue receives exactly one message on each input channel (can be understood as an implicit environment assumption) and outputs exactly one message along each output channel. The hardware queue is specified as follows.

HWQ — time_synchronous

in	$i : G_{Dlt};\ y : D_{Dlt}$
out	$o : D_{Dlt};\ x : G_{Dlt}$
asm	$Req\_Ok(i) \wedge FIFO\_Beh(x, y)$
com	$FIFO\_Beh(i, o) \wedge Req\_Ok(x) \wedge Bnd\_Hwm(i, y, o, x, W_h)$

The first conjunct of the assumption is the same as in FIFO_{TS}; the same holds for the first conjunct of the commitment. The second conjunct of the assumption implies that the hardware queue guarantees correct behaviour only as long as the software queue (composed with the interface component) behaves like a FIFO queue. The second conjunct of the commitment requires the hardware queue to use the software queue correctly, namely by sending requests only when the software queue has at least one data element to send in return. The third conjunct imposes the boundedness requirement on the number of messages that can be stored by the hardware queue: At any point in time, the number of data elements in the hardware queue is less than or equal to W_h, where W_h is some constant of type natural number. The auxiliary predicate *Bnd_Hwm* is formally defined as follows.

Bnd_Hwm

$a, b \in {G_{Dlt}}^{\infty};\ c, d \in {D_{Dlt}}^{\infty};\ t \in \mathbb{N}$

$\forall j \in \mathbb{N} : \#D\,Ⓢ\,(a{\downarrow_j}) + \#D\,Ⓢ\,(b{\downarrow_j}) - [\#D\,Ⓢ\,(c{\downarrow_j}) + \#D\,Ⓢ\,(d{\downarrow_j})] \leq t$

Interface Component The interface component communicates time-synchronously on the channels x and y. However, since the communication on the channels s and r is asynchronous, it cannot be specified in the time-synchronous format. As will be clear when we define specifications semantically in Sect. 4.2, time-synchronous communication cannot be captured in the time-independent format; this means that the time-dependent format is required.

The interface component is specified as follows.

INTF — time_dependent

in $x : G_{Dlt};\ s : D$

out $y : D_{Dlt};\ r : G$

asm $Time\_Synch(x) \wedge Req\_Ok(x)$

com $Eq(x, r, G) \wedge Time\_Synch(y) \wedge Eq(s, y, D)$

The keyword in the upper right corner specifies that INTF is time-dependent. In that case, the channel identifiers in the body represent timed infinite streams.

The first conjunct of the assumption restricts the input on the channel x to be time-synchronous; this is expressed by an auxiliary predicate defined formally as follows.

Time_Synch

$a \in M^{\underline{\infty}}$

$\forall j \in \mathbb{N} : \#M\,Ⓢ\,(a{\downarrow_j}) = j$

Due to this assumption, the input frequency on x is bounded — a fact which may simplify the implementation of the specification.

The task of the interface component is to convert the time-synchronous stream x into an ordinary timed stream r and to perform the opposite conversion with respect to s and y. This conversion can, of course, introduce additional delays. The second conjunct of the commitment makes sure that y is time-synchronous; the first and third conjunct employ an auxiliary predicate Eq to describe what it means to forward correctly. It is defined formally as follows.

Eq

$a, b \in M^{\infty/=};\ A \in \mathbb{P}(M)$

$A \circledS a = A \circledS b$

Thus, *Eq* requires that the streams a and b are identical when projected on the set of messages A. For any set S, $\mathbb{P}(S)$ yields the set $\{ T \mid T \subseteq S \}$.

Software Queue The specification of the software queue is expressed in terms of FIFO and substitution of channel identifiers:

$$\mathrm{SWQ} \equiv \mathrm{FIFO}[u \mapsto r, v \mapsto s]$$

The interpretation is as follows: The specification SWQ is equal to the specification obtained from FIFO by replacing its name by SWQ, the channel identifier u by r and the channel identifier v by s.

Composite Specification The three specifications HWQ, INTF and SWQ can be composed into a composite specification describing the network illustrated by Fig. 1 as follows.

FIFO_NET

in $i : G_{Dlt}$

out $o : D_{Dlt}$

loc $x : G_{Dlt};\ r : G;\ y : D_{Dlt};\ s : D$

$(o, x) := \mathrm{HWQ}(i, y) \quad (y, r) := \mathrm{INTF}(x, s) \quad (s) := \mathrm{SWQ}(r)$

The keyword loc distinguishes the declarations of the four local channels from the declarations of the external input and output channels. The body consists of the three component specifications introduced above represented as nondeterministic assignments with the output channels to the left and the input channels to the right. We require that the sub-specifications of a composite specification have disjoint sets of output identifiers. The semantics of composite specifications is defined formally in Sect. 4.3.

Both elementary and composite specifications are required to have disjoint sets of external input and output identifiers. In the composite case, the local channel identifiers must be different from the external channel identifiers. A composite specification is time-dependent, time-independent or time-synchronous depending on whether its elementary specifications are all time-dependent, time-independent or time-synchronous, respectively. Any other composite specification is *mixed*.

4 Refinement

The composite specification FIFO_NET is a conditional refinement of the elementary specification FIFO_{TS}, and FIFO_{TS} is a conditional refinement of the elementary specification FIFO. In this section we argue the correctness of this claim by mathematical means. For this purpose, we first describe a schematic translation of any specification into the time-dependent format; then we define the semantics of elementary and composite time-dependent specifications in a more mathematical manner and introduce two concepts of conditional refinement.

4.1 Schematic Translation into Time-Dependent Format

The time-independent and time-synchronous formats can be understood as syntactic sugar for the time-dependent format. In fact, any specification written in the time-independent or the time-synchronous format can be schematically translated into a by definition semantically equivalent time-dependent specification. For any time-independent elementary specification S, let $\mathbb{TD}(S)$ denote the time-dependent specification obtained from S by replacing the keyword time_independent by time_dependent and any occurrence of any channel identifier c in the body of S by $\overline{c}$. $\mathbb{TD}(S)$ captures the meaning of the time-independent specification S is a time-dependent setting.

Any elementary time-synchronous specification S can be translated into a by definition semantically equivalent time-dependent specification $\mathbb{TD}(S)$ by performing exactly the same modifications as in the time-independent case and, in addition, extending the assumption with the conjunct $\mathit{Time\_Synch}(i)$ for each input channel i, and the commitment with the conjunct $\mathit{Time\_Synch}(o)$ for each output channel o.

For any time-dependent elementary specification, we define $\mathbb{TD}(S) \equiv S$. If S is composite then $\mathbb{TD}(S)$ is equal to the result of applying $\mathbb{TD}$ to its component specifications.

For any elementary time-dependent specification S, by I_S, O_S, A_S and C_S we denote the set of typed input streams, the set of typed output streams, the assumption and the commitment of $\mathbb{TD}(S)$, respectively; we say that (I_S, O_S) is the *external interface* of S. If S is a composite specification we use I_S, O_S, L_S to denote the set of typed input, output and local streams of $\mathbb{TD}(S)$, respectively. For example, with respect to FIFO_NET of Sect. 3.3, we have

$$I_{\text{FIFO\_NET}} = \{i \in {G_{Dlt}}^{\underline{\infty}}\}$$

$$O_{\text{FIFO\_NET}} = \{o \in {D_{Dlt}}^{\underline{\infty}}\}$$

$$L_{\text{FIFO\_NET}} = \{x \in {G_{Dlt}}^{\underline{\infty}}, r \in G^{\underline{\infty}}, y \in {D_{Dlt}}^{\underline{\infty}}, s \in D^{\underline{\infty}}\}$$

Let V be a set of typed streams $\{v_1 \in T_1\underline{^\infty}, \ldots, v_n \in T_n\underline{^\infty}\}$ and P a formula. We define

$$\forall\, V : P \equiv \forall\, v_1 \in T_1\underline{^\infty};\ \ldots;\ v_n \in T_n\underline{^\infty} : P$$

$$\exists\, V : P \equiv \exists\, v_1 \in T_1\underline{^\infty};\ \ldots;\ v_n \in T_n\underline{^\infty} : P$$

4.2 Semantics of Elementary Time-Dependent Specifications

As already explained, there is a schematic translation of any time-independent or time-synchronous specification into the time-dependent format.

To capture the semantics of elementary time-dependent specifications, we introduce some helpful notational conventions. Let P be a formula whose free variables are among and typed in accordance with

$$V \equiv \{v_1 \in T_1\underline{^\infty}, \ldots, v_n \in T_n\underline{^\infty}\}$$

Hence, P is basically a predicate on timed infinite streams. By $P{\downarrow_j}$ we characterise the "prefix of P at time j". $P{\downarrow_j}$ is a formula whose free variables are among and typed in accordance with V. $P{\downarrow_j}$ holds if we can find extensions $v_1', \ldots, v_n'$ of $v_1{\downarrow_j}, \ldots, v_n{\downarrow_j}$ such that P' holds. Formally, for any $j \in \mathbb{N}_\infty$, we define $P{\downarrow_j}$ to denote the formula

$$\exists\, v_1' \in T_1\underline{^\infty};\ \ldots;\ v_n' \in T_n\underline{^\infty} : v_1{\downarrow_j} \sqsubseteq v_1' \wedge \ldots \wedge v_n{\downarrow_j} \sqsubseteq v_n' \wedge P'$$

where P' denotes the result of replacing each occurrence of v_j in P by v_j'. Note that $P{\downarrow_\infty} = P$. $P{\downarrow_\infty}$ has been introduced for the sake of convenience: It allows certain formulas, like the one below characterising the denotation of a specification, to be formulated more concisely. On some occasions we also need the prefix of P at time j with respect to a subset of free variables $A \subseteq V$; we define $P{\downarrow_{A:j}}$ to be equal to $P{\downarrow_j}$ if the free variables $V \setminus A$ are interpreted as constants.

The *denotation* $[\![\ S\ ]\!]$ of an elementary time-dependent specification S is defined by the formula:

$$\forall\, j \in \mathbb{N}_\infty : A_S{\downarrow_j} \Rightarrow C_S{\downarrow_{I_S:j}}{\downarrow_{O_S:j+1}}$$

Informally, S requires that:

partial input $(j < \infty)$: The output is in accordance with the commitment C_S until time $j+1$ if the input is in accordance with the assumption A_S until time j;

complete input $(j = \infty)$: The output is always in accordance with the commitment C_S if the input is always in accordance with the assumption A_S. Note that

$$(A_S{\downarrow_\infty} \Rightarrow C_S{\downarrow_{I_S:\infty}}{\downarrow_{O_S:\infty+1}}) \Leftrightarrow (A_S \Rightarrow C_S)$$

This one-unit-longer semantics, inspired by [1], requires a valid implementation to satisfy the commitment at least one time unit longer than the environment satisfies the assumption. This is basically a causality requirement: It disallows "implementations" that falsify the commitment because they "know" that the environment will falsify the assumption at some future point in time. Since no real implementation can predict the behaviour of the environment in this sense, we do not eliminate real implementations. Note that the assumption may refer to the output identifiers; this is often necessary to express the required assumptions about the input history (see HWQ in Sect. 3.3).

4.3 Semantics of Composite Time-Dependent Specifications

The denotation of a composite specification is defined in terms of the denotations of its component specifications. Let S be a composite specification whose body consists of m specifications

$$S_1, \ldots, S_m$$

Its denotation $[\![S]\!]$ is characterised by the formula

$$\exists L_S : [\![S_1]\!] \wedge \ldots \wedge [\![S_m]\!]$$

As explained in Sect. 4.1, L_S is the set of typed local streams in $\mathbb{TD}(S)$. To simplify the formal manipulation of composite specifications, we define $S_1 \otimes \ldots \otimes S_m$ to denote the composite specification S whose set of typed input, output and local streams are defined by

$$I_S \equiv (\cup_{j=1}^{m} I_{S_j}) \setminus (\cup_{j=1}^{m} O_{S_j})$$

$$O_S \equiv (\cup_{j=1}^{m} O_{S_j}) \setminus (\cup_{j=1}^{m} I_{S_j})$$

$$L_S \equiv (\cup_{j=1}^{m} I_{S_j}) \cap (\cup_{j=1}^{m} O_{S_j})$$

and whose body consists of the m specifications $S_1, .., S_m$ (represented in the form of nondeterministic assignments). For instance, FIFO_NET of Sect. 3.3 is equal to HWQ $\otimes$ INTF $\otimes$ SWQ.

It can be shown (see [6]) that if the component specifications are all realizable by functions that are contractive with respect to the Baire metric then the composite specification is also realizable with respect to such a function. Hence, when specifications are all realizable in this sense then there is at least one fixpoint when they are composed.

4.4 Two Concepts of Conditional Refinement

Consider two specifications S_1 and S_2, and a formula B whose free variables are all among (and typed in accordance with) $I_{S_2} \cup O_{S_2}$.

The specification S_2 is a *behavioural refinement* of the specification S_1 with respect to the *condition* B, written[1]

$$S_1 \rightsquigarrow_B S_2$$

if

$$I_{S_1} = I_{S_2}, \qquad O_{S_1} = O_{S_2}, \qquad \forall I_{S_1};\ O_{S_1} : B \wedge [\![S_2]\!] \Rightarrow [\![S_1]\!]$$

Behavioural refinement allows us to make additional assumptions about the environment since we consider only those input histories that satisfy the condition; moreover, it allows us to reduce under-specification since we only require that any input/output-history of S_2 is also an input/output-history of S_1, and not the other way around.

That the external interfaces of the two specifications are required to be the same is often inconvenient. We therefore also introduce a more general concept that characterises conditional refinement with respect to an interface translation (see Fig. 2). Assume that U and D are specifications such that

$$(I_{S_2} \setminus I_{S_1}) = I_U, \quad (I_{S_1} \setminus I_{S_2}) = O_U, \quad (O_{S_1} \setminus O_{S_2}) = I_D, \quad (O_{S_2} \setminus O_{S_1}) = O_D$$

The specification S_2 is an *interface refinement* of the specification S_1 with respect to the condition B, the *upwards relation* U and the *downwards relation* D, written

$$S_1 \overset{(U,D)}{\rightsquigarrow}_B S_2$$

if

$$U \otimes S_1 \otimes D \rightsquigarrow_B S_2$$

U translates input streams of S_2 to input streams of S_1; D translates output streams of S_1 to output streams of S_2 (in both cases, ignoring streams with identical names). Both the upwards and the downwards relations are defined as specifications, but these specifications are not to be implemented. Their task is to record design decisions with respect to the external interface.

Let DS (for dummy specification) represent the specification whose external interface is empty $(\{\}, \{\})$. We may then define behavioural refinement in terms of interface refinement as follows

$$S_1 \rightsquigarrow_B S_2 \equiv S_1 \overset{(\mathrm{DS},\mathrm{DS})}{\rightsquigarrow}_B S_2$$

We define the following short-hands

$$S_1 \rightsquigarrow S_2 \equiv S_1 \rightsquigarrow_{\mathsf{true}} S_2, \qquad S_1 \overset{(U,D)}{\rightsquigarrow} S_2 \equiv S_1 \overset{(U,D)}{\rightsquigarrow}_{\mathsf{true}} S_2$$

[1] In the same way as we distinguish between three formats for elementary specifications, we could also distinguish between three formats for conditions. $[\![\ ]\!]$ could then be overloaded to conditions in the obvious manner. However, to keep things simple, we view any condition as a formula whose free variables represent timed infinite streams.

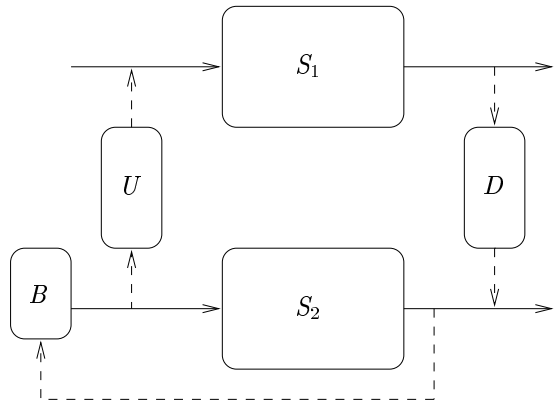


Fig. 2. Interface Refinement

Behavioural refinement is "transitive" in the following sense

$$S_1 \rightsquigarrow_{B_1} S_2 \wedge S_2 \rightsquigarrow_{B_2} S_3 \Rightarrow S_1 \rightsquigarrow_{B_1 \wedge B_2} S_3$$

Interface refinement satisfies a similar property (see App. A).

4.5 Correctness of the FIFO Decomposition

We first argue that FIFO_{TS} is an interface refinement of FIFO; thereafter, that FIFO_NET is a behavioural refinement of FIFO_{TS}. This allows us to deduce that FIFO_NET is an interface refinement of FIFO. We base our argumentation on deduction rules formulated in App. A and proved sound in App. B.

Since the external interfaces of FIFO and FIFO_{TS} are different, behavioural refinement is not sufficient; it is not just a question of renaming channels, we also have to translate time-synchronous streams with *Dlt* elements into streams without. The upwards and downwards relations are formally specified as follows.

U_{FIFO} — time_dependent

in	$i : G_{Dlt}$
out	$u : G$
com	$u = (G \cup \{\surd\}) \circledS\, i$

D_{FIFO} — time_dependent

in	$v : D$
out	$o : D_{Dlt}$
com	$v = (D \cup \{\surd\}) \circledS\, o$

Since in this particular case there are no assumptions to be imposed, the assumptions have been left out. It follows straightforwardly by the definition of behavioural refinement that

$$\mathrm{FIFO} \overset{(\mathrm{U_{FIFO}},\mathrm{D_{FIFO}})}{\rightsquigarrow}_{\mathrm{B_{FIFO}}} \mathrm{FIFO_{TS}} \tag{1}$$

where

$$\mathrm{B_{FIFO}} \equiv \mathit{Time\_Synch}(i)$$

The next step is to show that

$$\mathrm{FIFO_{TS}} \rightsquigarrow \mathrm{FIFO\_NET} \tag{2}$$

(2) is equivalent to

$$\mathrm{FIFO_{TS}} \rightsquigarrow \mathrm{HWQ} \otimes \mathrm{INTF} \otimes \mathrm{SWQ} \tag{3}$$

Let

$$\mathrm{FIFO'_{TS}} \equiv \mathrm{FIFO_{TS}}[i \mapsto x, o \mapsto y]$$

Since $\rightsquigarrow$ is reflexive, (2) follows by the transitivity and modularity rules (see App. A.1, A.2) if we can show that

$$\mathrm{FIFO_{TS}} \rightsquigarrow \mathrm{HWQ} \otimes \mathrm{FIFO'_{TS}} \tag{4}$$

$$\mathrm{FIFO'_{TS}} \rightsquigarrow \mathrm{INTF} \otimes \mathrm{SWQ} \tag{5}$$

Both (4) and (5) follow by the decomposition rule (see App. A.3). (1), (2) and the transitivity rule give

$$\mathrm{FIFO} \overset{(\mathrm{U_{FIFO}},\mathrm{D_{FIFO}})}{\rightsquigarrow}_{\mathrm{B_{FIFO}}} \mathrm{FIFO\_NET} \tag{6}$$

Thus, FIFO_NET is a conditional interface refinement of FIFO.

5 Imposing Additional Boundedness Constraints

The composite specification FIFO_NET does not impose constraints on the response time of the components: Replies can be issued after an unbounded delay. Moreover, the software queue was assumed to be unbounded. Since any computerised component has a bounded memory, this is not realistic. In fact, in FIFO_NET also the interface component is required to have an unbounded memory, since there is no upper bound on the message frequency for the channel s, and the communication on the channel y is time-synchronous. In this section, we show that the composite specification FIFO_NET can be refined into another composite specification $\mathrm{FIFO\_NET_B}$ in which response time constraints are imposed, and where additional environment assumptions make both the software queue and the interface component directly implementable. The response time constraints are informally described as follows:

- The hardware queue has a required delay of exactly T_h time units. Hence, it is required to reply to a request exactly T_h time units after the request is issued.
- The interface component has a required delay of not more than T_i time units when forwarding messages from the hardware queue to the software queue. Note that the delay can be less than T_i time units. Thus, the delay is not fixed as in the case of the hardware queue.
- The interface and the software queue together have a maximal delay of $T_s + 2 * T_i$ time units, where $*$ is the operator for multiplication. Any reply to a request made by the hardware queue is to be provided within this range of time. The software queue needs maximally T_s time units; the remaining $2 * T_i$ time units can be consumed by the interface.

T_h, T_i and T_s are all constants of type natural number. That the interface component and the software queue together have a maximal delay of $T_s + 2 * T_i$ time units does not mean that the interface component has a maximal delay of T_i time units when messages are forwarded from the software queue to the hardware queue. In fact, such a requirement would not be implementable. To see that, first note that the software queue may send an unbounded number of data elements within the same time unit. Assume, for example, that $T_i = 1$, and that the software queue sends three data elements in the nth time unit. Since the communication on y is time-synchronous, the interface component needs at least three time units to forward these messages along y thereby breaking the requirement that no data element should be delayed by more than one time unit. In fact, for the forwarding from the software queue to the hardware queue, the requirement below is sufficient:

- If at some point in time there are exactly e data elements that have been sent by the software queue, but not yet forwarded to the hardware queue, then the interface component will forward these data elements within the next $T_i + (e - 1)$ time units.

If the interface component satisfies this requirement then the hardware queue is guaranteed to receive a reply to each request within $T_s + 2 * T_i$ time units. To see that, first note that the communication on x is time-synchronous. Together with the timing constraints imposed on the communication along r and s this implies that if e messages are received within the same time unit on the channel s then not more than one of these, namely the first, can be received with the maximal delay of $T_i + T_s$ time units with respect to the corresponding request on x; the second cannot be delayed by more than $T_i + T_s - 1$ time units, and so on.

The constraint on the size of the internal memory is described informally as follows:

- The FIFO queue is not required to store more than W_f data elements, where W_f is a constant of type natural number. Since the hardware queue can store W_h data elements, this means that the software part does not have to store

more than $W_f - W_h$ data elements. Obviously, it holds that

$$W_h < W_f$$

Since the communication along x is time-synchronous, and the interface component has a maximal delay of T_i time units, the software queue will never receive more than T_i requests within one time unit on the channel r. This means that $T_i * T_s$ is an upper-bound on the number of data elements that can be sent by the software queue along the channel s within one time-unit.

5.1 Hardware Queue

The hardware queue is once more specified in the time-synchronous format. The only modification to the external interface is that the output channel o is renamed to w. This renaming is necessary since we want to translate o into w with the help of a downwards relation, and we require specifications (and thereby downwards relations) to have disjoint sets of input and output identifiers.

$\mathrm{HWQ_B}$ — time_synchronous

in	$i : G_{Dlt};\ y : D_{Dlt}$
out	$w : D_{Dlt};\ x : G_{Dlt}$
asm	$Req\_Ok(i) \wedge FIFO\_Beh(x, y)$
	$Bnd\_Rsp(x, y, T_s + 2 * T_i, \{Req\}, D) \wedge Bnd\_Qum(i, T_h, W_f)$
com	$Exact\_Rsp(i, w, T_h) \wedge Req\_Ok(x) \wedge Bnd\_Hwm(i, y, w, x, W_h)$
	$Bnd\_Qum(x, T_s + 2 * T_i, W_f - W_h)$

Throughout this paper: Line-breaks in assumptions and commitments represent logical conjunction. The two first conjuncts of the assumption have been inherited from HWQ; the same holds for the second and third conjunct of the commitment. Clearly, the hardware queue can only be required to fulfil its response time requirement as long as the two other components fulfil their response time requirements; the third conjunct of the assumption therefore requires the two environment components to reply to a request made by the hardware queue within $T_s + 2 * T_i$ time units. The auxiliary predicate *Bnd_Rsp* is formally defined as follows.

Bnd_Rsp

$a, b \in M^{\infty} \cup M^{\infty\!\!=};\ t \in \mathbb{N};\ A, B \in \mathbb{P}(M)$

$\forall\, j \in \mathbb{N} : \#[\, A \circledS (a{\downarrow_j})\,] \leq \#[\, B \circledS (b{\downarrow_{j+t}})\,]$

The fourth conjunct of the assumption makes sure that the FIFO queue is never required to store more than W_f data elements; the second parameter is needed since there is a delay of T_h time units between the transmission of a request and the output of the corresponding data element. The auxiliary predicate *Bnd_Qum* is formally defined as follows.

Bnd_Qum

$$a \in M^{\infty} \cup M^{\underline{\infty}};\ t, n \in \mathbb{N}$$

$$\forall j \in \mathbb{N} : \#[\, D \circledS (a{\downarrow_{j+t}})\,] - \#[\, \{Req\} \circledS (a{\downarrow_{j}})\,] \leq n$$

The fourth conjunct of the commitment formalises a similar requirement for the output along the channel x; note that this requirement is slightly stronger than it has to be since, for simplicity, we do not allow the hardware queue to exploit that the software part may reply in less than $T_s + 2 * T_i$ time units.

The first conjunct of the commitment represents both a strengthening and a weakening of the corresponding conjunct in HWQ. It employs an auxiliary predicate *Exact_Rsp* that is formally defined as follows.

Exact_Rsp

$$a \in {G_{Dlt}}^{\infty};\ b \in {D_{Dlt}}^{\infty};\ t \in \mathbb{N}$$

$$\forall j \in \mathbb{N}_+ : a.j = Req \Rightarrow b.(j+t) = (D \circledS a).(\#[\, \{Req\} \circledS (a{\downarrow_{j}})\,])$$

Exact_Rsp is a strengthening of *FIFO_Beh* in the sense that the reply is output with a delay of exactly t time units: If the jth message of the input stream a is a request then the $(j+t)$th message of the output stream b is the nth data element received on a, where n is the number of requests among the j first elements of a. *Exact_Rsp* is a weakening of *FIFO_Beh* in the sense that for any j such that $a.j \neq Req$, nothing is said about b at time $j+t$ except that the message output is an element of D_{Dlt}.

Correctness of Refinement Step Since the timed hardware queue may output an arbitrary data element in any time unit $j + T_h$ for which $i.j \neq Req$, it follows that $\mathrm{HWQ_B}$ is not a behavioural refinement of HWQ. However, we may find a downwards specification $\mathrm{D_{HWQ}}$, such that

$$\mathrm{HWQ} \overset{(\mathrm{DS},\mathrm{D_{HWQ}})}{\rightsquigarrow}_{\mathrm{B_{HWQ}}} \mathrm{HWQ_B} \tag{7}$$

where

$$\mathrm{B_{HWQ}} \equiv \mathit{Bnd\_Rsp}(x, y, T_s + 2 * T_i, \{Req\}, D) \wedge \mathit{Bnd\_Qum}(i, T_h, W_f)$$

$\mathrm{D_{HWQ}}$ can be defined as follows.

D_{HWQ} — time_synchronous

in	$o : D_{Dlt}$
out	$w : D_{Dlt}$
com	$\forall j \in \mathbb{N} : o.j \neq Dlt \Rightarrow w.j = o.j$

The correctness of (7) follows straightforwardly by the definition of interface refinement.

5.2 Interface Component

The interface component is once more specified in the time-dependent format.

$INTF_B$ — time_dependent

in	$x : G_{Dlt};\ s : D$
out	$y : D_{Dlt};\ r : G$
asm	$Time\_Synch(x) \wedge Req\_Ok(x) \wedge Bnd\_Frq(s, T_i * T_s)$
	$Bnd\_Qum(x, T_s + 2 * T_i, W_f - W_h)$
com	$Eq(x, r, G) \wedge Time\_Synch(y) \wedge Eq(s, y, D) \wedge Bnd\_Rsp(x, r, T_i, G, G)$
	$Weak\_Rsp(s, y, T_i) \wedge Bnd\_Qum(r, T_s, W_f - W_h)$

The two first conjuncts of the assumption and the three first conjuncts of the commitment restate INTF. The third conjunct of the assumption imposes a bound on the number of data elements that can be received during one time unit on s (see Page 16). The auxiliary predicate *Bnd_Frq* is defined as follows.

Bnd_Frq

$a \in M^{\underline{\infty}};\ t \in \mathbb{N}$

$\forall j \in \mathbb{N} : (\#\overline{a\downarrow_{j+1}} - \#\overline{a\downarrow_j}) \leq t$

Constraints similar to the fourth conjunct of the assumption and the sixth conjunct of the commitment have already been discussed in connection with HWQ_B. The fourth conjunct of the commitment requires that the conversion from x to r does not lead to a delay of more than T_i time units; the fifth imposes a weaker timing constraint on the communication in the other direction: If at some point in time there are exactly e data elements that have been sent on s but not

yet forwarded along y, then the interface component will forward these e data elements within the next $T_i + (e-1)$ time units.

the number of received data elements on the channel s is larger than the number of data elements sent on y, then at least one data element is forwarded along y within the next time unit (see explanation on Page 16). This requirement is formalised by the auxiliary predicate *Weak_Rsp* as follows.

Weak_Rsp

$a, b \in M^{\overline{\infty}};\ t \in \mathbb{N}$

$$\forall j, e \in \mathbb{N} : \#\overline{D \circledS a{\downarrow_j}} - \#\overline{D \circledS b{\downarrow_j}} = e \Rightarrow \#\overline{D \circledS b{\downarrow_{j+t+(e-1)}}} > \#\overline{D \circledS a{\downarrow_j}}$$

Correctness of Refinement Step Since the only difference between INTF and INTF_B is that both the assumption and the commitment have additional constraints, it follows straightforwardly that

$$\text{INTF} \rightsquigarrow_{\text{B}_\text{INTF}} \text{INTF}_\text{B} \tag{8}$$

where

$$\text{B}_\text{INTF} \equiv \mathit{Bnd\_Frq}(s, T_i * T_s) \wedge \mathit{Bnd\_Qum}(x, T_s + 2 * T_i, W_f - W_h)$$

5.3 Software Queue

The software queue is specified in the time-dependent format as follows.

SWQ_B — time_dependent

in $r : G$

out $s : D$

asm $\mathit{Req\_Ok}(r) \wedge \mathit{Bnd\_Qum}(r, T_s, W_f - W_h) \wedge \mathit{Bnd\_Frq}(r, T_i)$

com $\mathit{FIFO\_Beh}(r, s) \wedge \mathit{Bnd\_Rsp}(r, s, T_s, \{\mathit{Req}\}, D) \wedge \mathit{Bnd\_Frq}(s, T_i * T_s)$

SWQ_B differs from SWQ in that both the assumption and the commitment have additional conjuncts. The assumption has been strengthened to make sure that the software queue is never required to store more than $W_f - W_h$ data elements at the same point in time and never receives more than T_i messages within the same time unit. The additional conjuncts of the commitment require that the software queue replies to requests within T_s time units, and never sends more than $T_i * T_s$ messages along s within the same time unit.

Correctness of Refinement Step Since $\mathrm{SWQ_B}$ differs from SWQ only in that the assumption and the commitment have additional conjuncts, it follows trivially that

$$\mathrm{SWQ} \rightsquigarrow_{\mathrm{B_{SWQ}}} \mathrm{SWQ_B} \tag{9}$$

where

$$\mathrm{B_{SWQ}} \equiv \mathit{Bnd\_Qum}(r, T_s, W_f - W_h) \wedge \mathit{Bnd\_Frq}(r, T_i)$$

5.4 Composite Specification

The three elementary specifications presented above can be composed into a new composite specification as follows.

$\mathrm{FIFO\_NET_B}$

in	$i : G_{Dlt}$
out	$w : D_{Dlt}$
loc	$x : G_{Dlt};\ r : G;\ y : D_{Dlt};\ s : D$

$$(w, x) := \mathrm{HWQ_B}(i, y) \quad (y, r) := \mathrm{INTF_B}(x, s) \quad (s) := \mathrm{SWQ_B}(r)$$

Correctness of Refinement Step It must be shown that

$$\mathrm{FIFO} \overset{(\mathrm{U_{FIFO}}, \mathrm{D_{FIFO}} \otimes \mathrm{D_{HWQ}})}{\rightsquigarrow}_{\mathrm{B_{FIFO}} \wedge \mathrm{B_{FIFO}}'} \mathrm{FIFO\_NET_B} \tag{10}$$

where

$$\mathrm{B_{FIFO}}' \equiv \mathit{Bnd\_Qum}(i, T_h, W_f)$$

By (6) and the transitivity rule it is enough to show that

$$\mathrm{FIFO\_NET} \overset{(\mathrm{DS}, \mathrm{D_{HWQ}})}{\rightsquigarrow}_{\mathrm{B_{FIFO}}'} \mathrm{FIFO\_NET_B} \tag{11}$$

(11) follows by the modularity rule (see App. A.2).

6 Conclusions

This paper builds on earlier research, both by us and others: In particular, the notion of interface refinement is inspired by [3]; the notion of conditional refinement is investigated in [13]; similar concepts have been proposed by others (see for example [1]); the one-unit-longer semantics is adapted from [1]; the particular form of decomposition rule is discussed in [12]; the specification style has been taken from [4].

Specification languages based on the assumption/commitment paradigm have a long tradition. In fact, this style of specification was introduced with Hoare-logic [8]. The pre-condition of Hoare-logic can be thought of as an assumption about the initial state; the post-condition characterises a commitment any correct implementation must fulfil whenever the initial state satisfies the pre-condition. Well-known methods like VDM [10] and Z [11] developed from Hoare-logic. VDM employs the pre/post-style. In Z the pre-condition is stated implicitly and must be calculated. Together with the more recent B method [2], VDM and Z can be seen as leading techniques for the formal development of sequential systems. In the case of concurrency and nonterminating systems the assumption/commitment style of Hoare-logic is not sufficiently expressive. The paradigm presented in this paper is directed towards systems in which interaction and communication are essential features. Our approach is related to [1]. In contrast to [1] we work in the setting of streams and asynchronous message passing.

Conditional refinement is a flexible notion for relating specifications written at different levels of abstraction. Conditional refinement supports the introduction of boundedness constraints in specifications based on unbounded resources; in particular:

- Replacing purely asynchronous communication by time-synchronous communication.
- Replacing unbounded buffers by bounded buffers.
- Imposing additional boundedness constraints on the size of internal memories.
- Imposing additional boundedness requirements on the timing of input messages.

Conditional refinement is a straightforward extension or variant of well-known concepts for refinement [9, 10]. Traditional refinement relations typically allow the assumption to be weakened and the commitment to be strengthened modulo some translation of data structure. Conditional refinement allows both the assumption and the commitment to be strengthened; however, any strengthening of the assumption is recorded in a separate condition.

There are several alternative, not necessarily equivalent, ways to define conditional refinement. For instance, the directions of the upwards and downwards relations could be turned around. In that case we get the following definition of interface refinement

$$S_1 \overset{(D,U)}{\rightsquigarrow}_B S_2 \equiv S_1 \rightsquigarrow_B D \otimes S_2 \otimes U$$

where B is a constraint on the external interface of S_1. Alternatively, by allowing B to refer to the external interfaces of both S_1 and S_2 we could formulate the upwards and downwards relations within B. Which alternative is best suited from a pragmatic point of view is debatable.

According to [5], hardware/software co-design is the simultaneous design of both hardware and software to implement a desired function or specification. The presented approach should be well-suited to support this kind of design:

- It allows the integration of purely asynchronous, hand-shake (see [13]) and time-synchronous communication in the same composite specification.
- It allows specifications based on asynchronous communication to be refined into specifications based on time-synchronous communication, and vice-versa.

7 Acknowledgements

The authors have benefited from discussions with Manfred Broy on this and related topics.

References

1. M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17:507–533, 1995.
2. J.R. Abrial. *The B Book: Assigning Programs to Meaning.* Cambridge University Press, 1996.
3. M. Broy. Compositional refinement of interactive systems. Technical Report 89, Digital, SRC, Palo Alto, 1992.
4. M. Broy and K. Stølen. Focus on system development. Book manuscript, June 1998.
5. K. Buchenrieder, editor. *Third International Workshop on Hardware/Software Codesign.* IEEE Computer Society Press, 1994.
6. R. Grosu and K. Stølen. A model for mobile point-to-point data-flow networks without channel sharing. In *Proc. AMAST'96, Lecture Notes in Computer Science 1101*, pages 504–519, 1996.
7. O. Haugen. *Practitioners' Verification of SDL Systems.* PhD thesis, University of Oslo, 1997.
8. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.
9. C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–282, 1972.
10. C. B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, 1986.
11. J. M. Spivey. *Understanding Z, A Specification Language and its Formal Semantics.* Volume 3 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1988.
12. K. Stølen. Assumption/commitment rules for data-flow networks — with an emphasis on completeness. In *Proc. ESOP'96, Lecture Notes in Computer Science 1058*, pages 356–372, 1996.
13. K. Stølen. Refinement principles supporting the transition from asynchronous to synchronous communication. *Science of Computer Programming*, 26:255–272, 1996.
14. K. Stølen and P. Mohn. Measuring the effect of formalization. In *Proc. SAM'98*, Informatik Bericht Nr. 104, pages 183–190. Humboldt-Universität zu Berlin, 1998.

A Three Rules

In this paper we use several rules. Some of these rules are very simple and therefore not stated explicitly. For example, to prove (1) of Sect. 4.5 we need a rule capturing the definition of behavioural refinement. Three less trivial rules are formulated below; in App. B we prove their soundness. Any free variable is universally quantified over infinite timed streams of messages typed in accordance with the corresponding channel declaration.

A.1 Transitivity Rule

$$\frac{\begin{array}{l} B \wedge [\![\ U_2\ ]\!] \wedge [\![\ D_2\ ]\!] \Rightarrow B_1 \\ B \Rightarrow B_2 \\ [\![\ U_1\ ]\!] \wedge [\![\ U_2\ ]\!] \Rightarrow [\![\ U\ ]\!] \\ [\![\ D_1\ ]\!] \wedge [\![\ D_2\ ]\!] \Rightarrow [\![\ D\ ]\!] \\ S_1 \overset{(U_1,D_1)}{\rightsquigarrow}_{B_1} S_2 \\ S_2 \overset{(U_2,D_2)}{\rightsquigarrow}_{B_2} S_3 \end{array}}{S_1 \overset{(U,D)}{\rightsquigarrow}_{B} S_3}$$

Some intuition:

- Premises 1 and 2 make sure that the overall condition B is stronger than B_1 and B_2.
- Premise 3 makes sure that the upwards relation obtained by connecting U_2 and U_1 is allowed by U.
- Premise 4 makes sure that the downwards relation obtained by connecting D_1 and D_2 is allowed by D.
- Premises 5 and 6 are illustrated by Fig. 3.

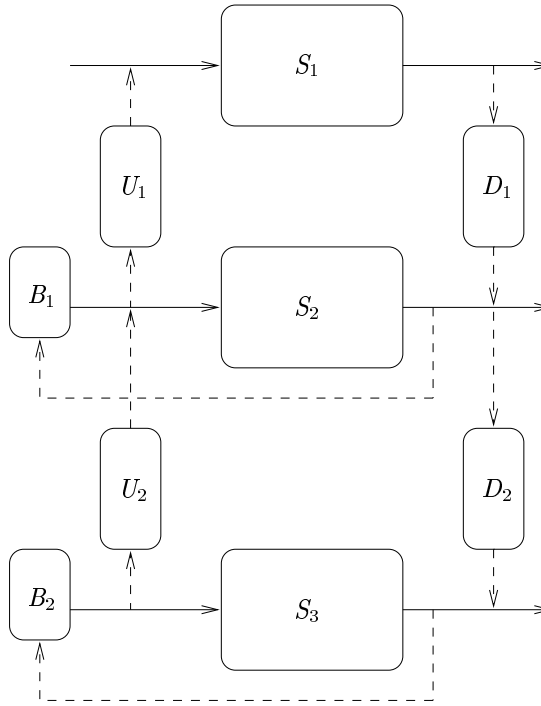


Fig. 3. Illustration of the Fifth and Sixth Premise in the Transitivity Rule

A.2 Modularity Rule

$$\frac{\begin{array}{l}\wedge_{i=1}^{n}(B_i \Rightarrow !\, O_{U_i} : [\![\ U_i\ ]\!]) \\ \exists\, I_{\otimes_{i=1}^{n} S_i};\ L_{\otimes_{i=1}^{n} S_i};\ O_{\otimes_{i=1}^{n} S_i} : \wedge_{i=1}^{n}([\![\ U_i\ ]\!] \wedge [\![\ D_i\ ]\!]) \\ (\wedge_{i=1}^{n}[\![\ U_i\ ]\!]) \Rightarrow [\![\ U\ ]\!] \\ (\wedge_{i=1}^{n}[\![\ D_i\ ]\!]) \Rightarrow [\![\ D\ ]\!] \\ B \wedge (\wedge_{i=1}^{n}[\![\ S_i'\ ]\!]) \Rightarrow \wedge_{i=1}^{n} B_i \\ \wedge_{i=1}^{n}(S_i \overset{(U_i, D_i)}{\rightsquigarrow}_{B_i} S_i')\end{array}}{\otimes_{i=1}^{n} S_i \overset{(U, D)}{\rightsquigarrow}_{B} \otimes_{i=1}^{n} S_i'}$$

$!\, V : P$ holds if there is a unique V such that P holds. Some intuition:

- Premise 1 makes sure that for each concrete input history such that B_i holds, the upwards relation U_i allows exactly one abstract input history. Hence, each concrete input history that satisfies the condition is related to exactly one abstract input history, but the same abstract input history can be related to several concrete input histories.
- Premise 2 makes sure that the conjunction of the downwards and upwards relations is consistent.
- Premise 3 makes sure that the upwards relation described by the conjunction of the n upwards relations U_i is allowed by the overall upwards relation U.
- Premise 4 makes sure that the upwards relation described by the conjunction of the n downwards relations D_i is allowed by the overall downwards relation D.
- Premise 5 makes sure that the assumptions made by the n B_i's are fulfilled by the composite specification consisting of the n specifications S_i' when the input from the overall environment satisfies B.

A.3 Decomposition Rule

$$\frac{\begin{array}{l}A \Rightarrow A_1{\downarrow_0} \wedge A_2{\downarrow_0} \\ \forall\, j \in \mathbb{N} : A \wedge C_1{\downarrow_{I_1:j}}{\downarrow_{O_1:j+1}} \wedge C_2{\downarrow_{I_2:j}}{\downarrow_{O_2:j+1}} \Rightarrow A_1{\downarrow_{j+1}} \wedge A_2{\downarrow_{j+1}} \\ A \wedge \langle C_1 \rangle \wedge \langle C_2 \rangle \Rightarrow (A_1 \wedge (C_1 \Rightarrow A_2)) \vee (A_2 \wedge (C_2 \Rightarrow A_1)) \\ \forall\, j \in \mathbb{N}_\infty : A{\downarrow_j} \wedge C_1{\downarrow_{I_1:j}}{\downarrow_{O_1:j+1}} \wedge C_2{\downarrow_{I_2:j}}{\downarrow_{O_2:j+1}} \Rightarrow C{\downarrow_{I:j}}{\downarrow_{O:j+1}}\end{array}}{S \rightsquigarrow S_1 \otimes S_2}$$

$\langle P \rangle$ denotes the upwards closure of P; formally $\langle P \rangle \equiv \forall j \in \mathbb{N} : P{\downarrow_j}$. S is a specification with assumption A, commitment C, and external interface (I, O). As illustrated by Fig. 4, the relationship between S_1/S_2 and

$$A_1, C_1, (I_1, O_1)/A_2, C_2, (I_2, O_2)$$

is defined accordingly. As explained in detail in App. B.3, the correctness of the decomposition rule follows by induction. Some intuition:

- Premise 1 makes sure that the component assumptions A_1 and A_2 hold at time 0.
- Premise 2 makes sure that the component assumptions A_1 and A_2 hold at time $j+1$ if the component commitments C_1 and C_2 hold at time $j+1$.
- Premise 3 is concerned with liveness in the assumptions. This premise is not required if both A_1 and A_2 are upwards closed.
- Premise 4 makes sure that the overall commitment C holds at time $j+1$ if the component commitments C_1 and C_2 hold at time $j+1$.

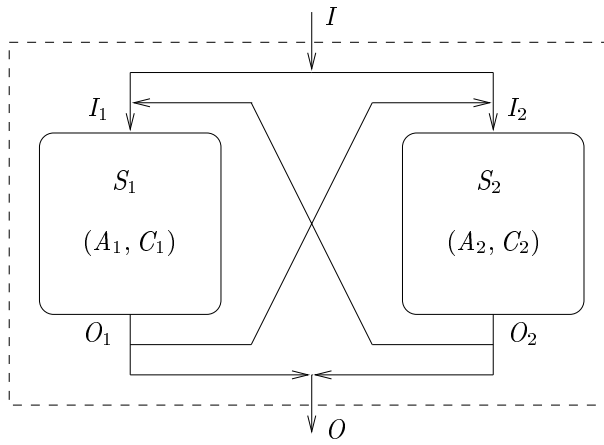


Fig. 4. Network Represented by $S_1 \otimes S_2$

B Soundness of Rules

In this appendix we prove that the three rules formulated in App. A are sound. Any free variable is universally quantified over infinite timed streams of messages typed in accordance with the corresponding channel declaration.

B.1 Transitivity Rule

Given

$$B \wedge [\![\ U_2\ ]\!] \wedge [\![\ D_2\ ]\!] \Rightarrow B_1 \tag{12}$$

$$B \Rightarrow B_2 \tag{13}$$

$$[\![\ U_1\ ]\!] \wedge [\![\ U_2\ ]\!] \Rightarrow [\![\ U\ ]\!] \tag{14}$$

$$[\![\ D_1\ ]\!] \wedge [\![\ D_2\ ]\!] \Rightarrow [\![\ D\ ]\!] \tag{15}$$

$$S_1 \overset{(U_1, D_1)}{\rightsquigarrow}_{B_1} S_2 \tag{16}$$

$$S_2 \overset{(U_2, D_2)}{\rightsquigarrow}_{B_2} S_3 \tag{17}$$

It must be shown that

$$S_1 \overset{(U, D)}{\rightsquigarrow}_{B} S_3 \tag{18}$$

(18) is equivalent to

$$B \wedge [\![\ S_3\ ]\!] \Rightarrow [\![\ U \otimes S_1 \otimes D\ ]\!] \tag{19}$$

To prove (19), assume there are I_{S_3}, O_{S_3} such that

$$B \tag{20}$$

$$[\![\ S_3\ ]\!] \tag{21}$$

(13), (20) imply

$$B_2 \tag{22}$$

(17), (21), (22) imply

$$[\![\ U_2 \otimes S_2 \otimes D_2\ ]\!] \tag{23}$$

(23) implies there are O_{U_2}, I_{D_2} such that

$$[\![\ U_2\ ]\!] \tag{24}$$

$$[\![\ S_2\ ]\!] \tag{25}$$

$$[\![\ D_2\ ]\!] \tag{26}$$

(12), (20), (24), (26) imply

$$B_1 \tag{27}$$

(16), (25), (27) imply

$$[\![\ U_1 \otimes S_1 \otimes D_1\ ]\!] \tag{28}$$

(28) implies there are O_{U_1}, I_{D_1} such that

$$[\![\ U_1\ ]\!] \tag{29}$$

$$[\![\ S_1\ ]\!] \tag{30}$$

$$[\![\ D_1\ ]\!] \tag{31}$$

(14), (24), (29) imply

$$[\![\ U\ ]\!] \tag{32}$$

(15), (26), (31) imply

$$[\![\ D\ ]\!] \tag{33}$$

(30), (32), (33) imply

$$[\![\ U \otimes S_1 \otimes D\ ]\!] \tag{34}$$

The way (34) was deduced from (20), (21) implies (19).

B.2 Modularity Rule

Given

$$\wedge_{i=1}^{n}(B_i \Rightarrow !O_{U_i} : [\![\ U_i\ ]\!]) \tag{35}$$

$$\exists\, I_{\otimes_{i=1}^{n} S_i};\ L_{\otimes_{i=1}^{n} S_i};\ O_{\otimes_{i=1}^{n} S_i} : \wedge_{i=1}^{n}([\![\ U_i\ ]\!] \wedge [\![\ D_i\ ]\!]) \tag{36}$$

$$(\wedge_{i=1}^{n}[\![\ U_i\ ]\!]) \Rightarrow [\![\ U\ ]\!] \tag{37}$$

$$(\wedge_{i=1}^{n}[\![\ D_i\ ]\!]) \Rightarrow [\![\ D\ ]\!] \tag{38}$$

$$B \wedge (\wedge_{i=1}^{n}[\![\ S_i'\ ]\!]) \Rightarrow \wedge_{i=1}^{n} B_i \tag{39}$$

$$\wedge_{i=1}^{n}(S_i \overset{(U_i,D_i)}{\rightsquigarrow}_{B_i} S_i') \tag{40}$$

It must be shown that

$$\otimes_{i=1}^{n} S_i \overset{(U,D)}{\rightsquigarrow}_{B} \otimes_{i=1}^{n} S_i' \tag{41}$$

(41) follows if we can show that

$$B \wedge [\![\ \otimes_{i=1}^{n} S_i'\ ]\!] \Rightarrow [\![\ U \otimes (\otimes_{i=1}^{n} S_i) \otimes D\ ]\!] \tag{42}$$

To prove (42), assume there are $I_{\otimes_{i=1}^{n} S_i'}, L_{\otimes_{i=1}^{n} S_i'}, O_{\otimes_{i=1}^{n} S_i'}$ such that

$$B \tag{43}$$

$$\wedge_{i=1}^{n}[\![\ S_i'\ ]\!] \tag{44}$$

(39), (43), (44) imply

$$\wedge_{i=1}^{n} B_i \tag{45}$$

(40), (44), (45) imply

$$\wedge_{i=1}^{n} [\![\ U_i \otimes S_i \otimes D_i\ ]\!] \tag{46}$$

(46) implies

$$\wedge_{i=1}^{n} \exists\, O_{U_i}, I_{D_i} : [\![\ U_i\ ]\!] \wedge [\![\ S_i\ ]\!] \wedge [\![\ D_i\ ]\!] \tag{47}$$

(35), (36), (47) imply there are $I_{\otimes_{i=1}^{n} S_i}$, $L_{\otimes_{i=1}^{n} S_i}$, $O_{\otimes_{i=1}^{n} S_i}$ such that

$$\wedge_{i=1}^{n} [\![\ U_i\ ]\!] \tag{48}$$

$$\wedge_{i=1}^{n} [\![\ S_i\ ]\!] \tag{49}$$

$$\wedge_{i=1}^{n} [\![\ D_i\ ]\!] \tag{50}$$

(37), (38), (48), (50) imply

$$[\![\ U\ ]\!] \tag{51}$$

$$[\![\ D\ ]\!] \tag{52}$$

(49), (51), (52) imply

$$[\![\ U \otimes (\otimes_{i=1}^{n} S_i) \otimes D\ ]\!] \tag{53}$$

The way (53) was deduced from (43), (44) implies (42).

B.3 Decomposition Rule

Given

$$A \Rightarrow A_1{\downarrow_0} \wedge A_2{\downarrow_0} \tag{54}$$

$$\forall\, j \in \mathbb{N} : A \wedge C_1{\downarrow_{I_1:j}}{\downarrow_{O_1:j+1}} \wedge C_2{\downarrow_{I_2:j}}{\downarrow_{O_2:j+1}} \Rightarrow A_1{\downarrow_{j+1}} \wedge A_2{\downarrow_{j+1}} \tag{55}$$

$$A \wedge \langle C_1 \rangle \wedge \langle C_2 \rangle \Rightarrow (A_1 \wedge (C_1 \Rightarrow A_2)) \vee (A_2 \wedge (C_2 \Rightarrow A_1)) \tag{56}$$

$$\forall\, j \in \mathbb{N}_\infty : A{\downarrow_j} \wedge C_1{\downarrow_{I_1:j}}{\downarrow_{O_1:j+1}} \wedge C_2{\downarrow_{I_2:j}}{\downarrow_{O_2:j+1}} \Rightarrow C{\downarrow_{I:j}}{\downarrow_{O:j+1}} \tag{57}$$

It must be shown that

$$S \rightsquigarrow S_1 \otimes S_2 \tag{58}$$

(58) is equivalent to

$$[\![\ S_1 \otimes S_2\ ]\!] \Rightarrow [\![\ S\ ]\!] \tag{59}$$

(59) is equivalent to

$$\begin{aligned} &(\forall\, j \in \mathbb{N}_\infty : A_1{\downarrow_j} \Rightarrow C_1{\downarrow_{I_1:j}}{\downarrow_{O_1:j+1}}) \wedge \\ &(\forall\, j \in \mathbb{N}_\infty : A_2{\downarrow_j} \Rightarrow C_2{\downarrow_{I_2:j}}{\downarrow_{O_2:j+1}}) \\ &\qquad \Rightarrow \\ &(\forall\, j \in \mathbb{N}_\infty : A{\downarrow_j} \Rightarrow C{\downarrow_{I:j}}{\downarrow_{O:j+1}}) \end{aligned} \tag{60}$$

(60) is equivalent to

$$\begin{aligned}
&\forall j \in \mathbb{N}_\infty : \\
&\quad A{\downarrow_j} \wedge \\
&\quad (\forall j \in \mathbb{N}_\infty : A_1{\downarrow_j} \Rightarrow C_1{\downarrow_{I_1:j}}{\downarrow_{O_1:j+1}}) \wedge \\
&\quad (\forall j \in \mathbb{N}_\infty : A_2{\downarrow_j} \Rightarrow C_2{\downarrow_{I_2:j}}{\downarrow_{O_2:j+1}}) \\
&\qquad \Rightarrow \\
&\quad C{\downarrow_{I:j}}{\downarrow_{O:j+1}}
\end{aligned} \tag{61}$$

To prove (61), assume there are $I, L_{S_1 \otimes S_2}, O, l$ such that

$$A{\downarrow_l} \tag{62}$$

$$\forall j \in \mathbb{N}_\infty : A_1{\downarrow_j} \Rightarrow C_1{\downarrow_{I_1:j}}{\downarrow_{O_1:j+1}} \tag{63}$$

$$\forall j \in \mathbb{N}_\infty : A_2{\downarrow_j} \Rightarrow C_2{\downarrow_{I_2:j}}{\downarrow_{O_2:j+1}} \tag{64}$$

There are two cases to consider. Assume

$$l < \infty \tag{65}$$

(54), (55), (62), (63), (64) and induction on j imply

$$j \leq l \Rightarrow A_1{\downarrow_j} \wedge A_2{\downarrow_j} \tag{66}$$

(63), (64), (66) imply

$$C_1{\downarrow_{I_1:l}}{\downarrow_{O_1:l+1}} \wedge C_2{\downarrow_{I_2:l}}{\downarrow_{O_2:l+1}} \tag{67}$$

(57), (62), (67) imply

$$C{\downarrow_{I:l}}{\downarrow_{O:l+1}} \tag{68}$$

The way (68) was deduced from (62), (63), (64) proves (61) for the case that (65) holds. Assume

$$l = \infty \tag{69}$$

By the same inductive argument as above

$$\langle C_1 \rangle \wedge \langle C_2 \rangle \tag{70}$$

(56), (62), (69), (70) imply

$$(A_1 \wedge (C_1 \Rightarrow A_2)) \vee (A_2 \wedge (C_2 \Rightarrow A_1)) \tag{71}$$

(63), (64), (71) imply

$$C_1 \wedge C_2 \tag{72}$$

(57), (62), (69), (72) imply

$$C \tag{73}$$

The way (73) was deduced from (62), (63), (64) proves (61) for the case that (69) holds.