Refinement of Time[1)]

Manfred Broy
Institut für Informatik
Technische Universität München
80290 München, Germany

Abstract

We introduce a mathematical model of the timed behaviour of components with streams as input and output using a hierarchy of timing concepts. We distinguish *non-timed streams*, *discrete streams* with *discrete* or with *continuous time*, and *dense streams with continuous time*. We introduce a notion of a timed system component and formulate requirements for the time flow. We show how to compose timed systems in a modular way. We show that the introduction of time into a system model as well as the change of the timing model in the system development process is a refinement step.

1. Introduction

Although time is an important issue for many information processing systems all the early attempts to provide logical, algebraic, or mathematical foundations for programming and system development tried to abstract entirely from timing issues. This is of course fine as long as we are only interested in sequential algorithms. However, looking at interactive systems, especially at reactive embedded systems, timing issues become crucial. The same holds for many application systems of today that have to react to time events. This is why we need well worked out timing models that are well suited for the more abstract techniques for the specification, verification, and refinement of systems.

A descriptive functional semantic model of distributed systems of concurrently interacting components is of major interest in many research areas and applications of computing science and systems engineering. For the modular systematic development of information processing systems we need precise and readable interface descriptions of system components. We require that such interface descriptions document all information about the syntactic and semantic properties of a component that we need to use it properly. In an interface specification we also describe the time requirements and dependencies in the behaviour of a component.

We are interested in the description of components that react interactively to input by output. Both input and output take place within a global time frame. The modular specification of the observable behaviour of interactive systems is an important technique in system and software development. We speak of *black box specifications* or *interface specifications*. An adequate concept of interface specification does not only depend on a

[1)] This work was partially sponsored by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen", by the BMBF project KorSys, and the industrial research project SysLab.

simple notion of observability, but also on the operators that we apply to compose components into systems.

Although ignored in theoretical computer science for a while, the incorporation of time and its formal representation is of essential interest for system models. In time dependent systems, the timing and the data values of the output depend upon the timing and the data values of the input. However, for certain components the timing of the input does not influence the data values of the output, but only their timing. Often then the timing is rather unimportant. In these cases, we can describe the input/output behaviour of a component without explicit reference to time.

It is one of the goals of this paper to show what consequences the introduction of time in a semantic model actually has. The semantics gets more robust since the flow of time leads to a explicit modelling of causality and thus to more realistic models of computations. As a consequence, fixpoint theory gets more straightforward and does not need any sophisticated theoretical concepts. This simplicity is lost, however, if we abstract away timing information completely or partially as it is done, for instance, in models of computations based on the idea of full synchrony. Here the lack of explicit causality leads into semantic pathologies

In the following, we introduce a semantic model of interface behaviour that includes discrete and continuous streams with discrete and continuous time and study composition operators. On the basis of this semantic model we introduce more pragmatic specification techniques for the description of reactive components.

In a first section, we introduce our mathematical basis. Then we show how to describe the syntactic interfaces and the dynamic behaviours of interactive systems. We treat composition operators and introduce concepts of refinement for timing issues.

2. Streams

In this section we introduce the basic mathematical concept of streams. A stream describes the communication history of a channel or the flow of values assumed by a variable of a system.

Roughly speaking a stream is a finite or an infinite sequence or flow of values from a set M called the *sort* of the stream. If additional time information is contained, we speak of a *timed stream*. In the following, we are interested to separate aspects of data and message flow of a channel on variable from timing aspects. We define four concepts of streams: non-timed streams and discrete streams with discrete and continuous time, and finally dense streams with continuous time.

2.1 Non-timed Discrete Streams

Given a set M of messages a discrete stream over the set M is a finite or infinite sequence of elements from M. By M^* we denote the finite sequences over the set M. The set M^* includes the empty sequence that we denote by ‹›. By s^r we denote the concatenation of two sequences s and r.

By M^∞ (we write M^∞ for the function space $\mathbb{N}^+ \to M$ where $\mathbb{N}^+$ stands for $\mathbb{N} \setminus \{0\}$) we denote the set of infinite sequences over the set M. By M^ω we denote the set of non-timed streams. It is defined by

$$M^{\omega} = M^{\infty} \cup M^{*}$$

Non-timed streams do not contain any information about the timing of the messages in the stream. By s.i we denote the i-th message in a stream. By #s we denote the length of the stream s which is an element in the set $\mathbb{N} \cup \{\infty\}$.

2.2 Time

Time can be modelled in many ways. Typical time models that we find in the literature are the natural numbers $\mathbb{N}$ and the positive real numbers $\mathbb{R}^{+}$. We might also work with the rational numbers, however, there is perhaps not a significant difference between the real numbers and the rational numbers when modelling time. These numbers are all models of linear time. Linear time is most appropriate for models of global time.

A different concept is *distributed* time[2]. Distributed time helps to model systems with distributed components and local clocks that run with independent speed. We distinguish between *physical* time (which is global) and *system* time represented by local or global clocks. It is difficult to speak about distributed time without an explicit or implicit notion of global time. This way we get a layered time model. The base layer is global time. The next layer is system time, that corresponds to local clocks specified in terms of the global time.

We are in the following mainly interested in global time models and concentrate therefore on linear time models.

A central notion in this paper is discreteness of time in contrast to continuous time. There are several ways to define discreteness of a set by mathematical methods using notions from topology, metric spaces, or partial orderings. We work only with two sets for representing time namely the natural numbers for discrete time and real numbers for continuous time.

In the following we distinguish between discrete time (time is represented by the elements from a discrete set) and discrete message sets (the messages are taken from a set that is discrete). Therefore it makes sense to speak of time-discrete in contrast to time-continuous streams and also of message-discrete in contrast to message-continuous streams.

2.3 Discrete Streams with Discrete Time

A timed communication history for a channel carrying messages from a given set M is represented by a timed stream. A timed stream with discrete time is a finite or infinite sequence of messages with additional timing information from a discrete time space. We work with the following notions of discrete time.

- Discrete time: A discrete stream with discrete time s with #s messages is a finite or infinite sequence of messages with a timing function

$$s^{™}: [1 : 1+\#s[\rightarrow \mathbb{N}$$

---

[2] In temporal logic the notions of linear versus branching time are used. Branching time, however, is used to model different nondeterministic choices for system executions - not distributed time.

which is weakly monotonic. $s\text{™}i$ associates with the i-th message in s its time stamp.

A special case is a stream s with

$$s\text{™}i = i \qquad \text{for all } i \in [1{:}1+\#s[$$

Such a stream is called *time synchronous* since it issues exactly one message at a time. This is a good model for hardware systems with a global clock pulse. A further special case is a stream where the timing function is strictly monotonic. This models the case where at most one message is communicated at each time point. In other words, the time granularity is chosen fine enough that all messages are separated by the time scale.

Formally we can represent the set of streams with discrete time by the set

$$M^{\aleph} = (M^*)^{\infty}.$$

For $s \in M^{\aleph}$ we denote the number of messages in s by #s. By s:i we denote the sequence of messages in the i-th time interval. We denote for $i \in \mathbb{N}$, $s \in M^{\aleph}$ by

$$s{\downarrow}i$$

the sequence of the first i sequences in the stream s. Thus $s{\downarrow}i$ denotes the communication history of the stream s for the first i time intervals. By

$$\overline{s}$$

we denote the finite or infinite stream that is the result of concatenating all sequences in the stream s. This corresponds to a time abstraction in which we forget all the timing information in a stream.

We use both notations for time abstraction and the restriction of streams up to a time point also for tuples and sets of timed streams by applying them pointwise. We define the timing function $s\text{™}$ for $s \in M^{\aleph}$ simply by (let $i \in [1{:}\ 1+\#s[$):

$$s\text{™}i = \min \{j \in \mathbb{N}{:}\ i \le \#(s{\downarrow}j)\}$$

Note that the set $M^{\aleph}$ is isomorphic to the set of streams over the set $M \cup \{\surd\}$ with an infinite number of time ticks.

2.4 Discrete Streams with Continuous Time

Continuous time is similar to discrete time as long as we work with discrete events (sampled message streams).

- Real time (continuous time) with discrete streams (*sampling*): We can also choose a timing by real numbers associated with a stream s. Then the timing is described by the function:

  $$s\text{™}: [1 : 1+\#s[\to \mathbb{R}^+$$

  Again $s\text{™}i$ associates a time point with the i-th message in the stream s. Here the time points can be chosen more freely. Here we require strict monotonicity since continuous time is the finest time granularity we can choose.

The set of discrete streams over the message set M with continuous time is represented by the set

$$M^{\Re}$$

In this model of timed communication we have to cope with Zeno`s paradox. We may have

$$\forall\, i \in [1 : 1+\#s[: s^{\tm} i < t$$

for some time $t \in \mathbb{R}$, although $\#s = \infty$. In many applications such a behaviour should be excluded. A simple way to do this is to assume a minimal time distance $\delta \in \mathbb{R}$, $\delta > 0$, for all the messages in s such that

$$s^{\tm}(i+1) - s^{\tm} i > \delta \text{ for all } i \in [1 : \#s[.$$

For a stream s we denote by for $t \in \mathbb{R}^+$

$$s{\downarrow}t$$

the stream of messages till time point t.

2.5 Dense Streams

So far the streams were discrete in the sense that a discrete stream always is a sequence of messages where we could speak about the first, second, third, and so on message in a stream. Using continuous time a stream may contain uncountably many message elements. We speak of a *dense stream.*

- A dense stream is represented by a function

  $$s: \mathbb{R}^+ \to M$$

  For every point in time $t \in \mathbb{R}^+$ we obtain a message $s(t) \in M$. If we include a dummy message $\blacklozenge \in M$ we even may represent all other types of timed streams by dense streams s where the set of time points with actual messages

  $$\{t \in \mathbb{R}^+: s(t) \neq \blacklozenge\}$$

  is finite or at least countable. By

  $$M^{\mathbb{R}}$$

  we denote the set of dense streams.

We easily may extend the notation $s{\downarrow}t$ to dense streams. We extend this notation of truncating streams at time points to sets of streams $W \subseteq M^{\mathbb{R}}$ pointwise

$$W{\downarrow}t = \{s{\downarrow}t: s \in W\}$$

It does not make sense to extend the time abstraction or the length function to dense streams. We can, however, work with sampling. Moreover, we can embed discrete streams into dense streams.

We call a dense stream $s \in M^{\mathbb{R}}$ *quasidiscrete*, if there exists a sequence of time points t_i, $i \in \mathbb{N}$, such that the set $\{t_i: i \in \mathbb{N}\}$ is unbounded, mathematically expressed by

$$\forall\, j \in \mathbb{N}: \exists\, i \in \mathbb{N}: j \leq t_i$$

and the stream s restricted to the time intervals $[t_i : t_{i+1}[$ is a constant function, formally

$$\forall\, i \in \mathbb{N}: \exists\, m \in M: \forall\, t \in [t_i : t_{i+1}[: s.t = m$$

Then s changes its values only at countably many time points. Quasidiscrete streams can easily be represented by discrete streams by recording the changes at the time points t_i only.

We distinguish between discrete streams with continuous time and dense streams with continuous time. In time continuous systems we work with continuous time. In addition, the message set may be non-discrete. The classical case from mathematics where continuous functions on real numbers are considered is the certainly best known example of a dense stream with continuous time and continuously changing messages. Of course, this theory of continuous real-valued function can be generalised to any continuous message set along the work in metric spaces and topology.

3. Components as Functions on Streams

In this section we introduce a general concept of a component as a function on time streams. We consider the most general case of dense streams.

3.1 Behaviours of Components

Given a set C of sorted identifiers we denote by

$$\vec{C}$$

the set of valuations

$$x: C \to M^{\mathbb{R}}$$

where x.c is a time stream of the appropriate sort of channel $c \in C$. A function

$$F: \vec{I} \to \wp(\vec{O})$$

is called a *component behaviour*. F is called

- *timed* or *weakly causal*, if for all $t \in \mathbb{R}$ we have

  $$x{\downarrow}t = z{\downarrow}t \Rightarrow F(x){\downarrow}t = F(z){\downarrow}t$$

- *time guarded by a finite delay* $\delta \in \mathbb{R}^+$, $\delta > 0$, or *causal* if for all $t \in \mathbb{R}$ we have

  $$x{\downarrow}t = z{\downarrow}t \Rightarrow F(x){\downarrow}(t+\delta) = F(z){\downarrow}(t+\delta)$$

- *realisable*, if there exists a time guarded function $f: \vec{I} \to \vec{O}$ (with some finite delay δ) such that for all input histories x:

  $$f.x \in F.x$$

We use time guarded stream processing function F to model the behaviour of a component. A graphic representation of the function F is given in Fig. 1.

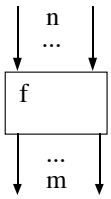

Fig. 1 Graphical representation of a component with n input and m output channels

A component function F is called

- time independent, if for discrete input histories x and z:

$$\overline{x} = \overline{z} \Rightarrow \overline{F.x} = \overline{F.z}$$

For time independent behaviours the timing of the messages in the input streams does not influence the messages in the output streams but only their timing. We generalise this definition to functions on dense streams by (monotonic) time transformations.

A time transformation Θ is a monotonic total function

$$\Theta : \mathbb{R}^+ \to \mathbb{R}^+$$

that is unbounded. Here monotonicity means:

$$\forall\, t, r \in \mathbb{R}^+: t \le r \Rightarrow \Theta(t) \le \Theta(r)$$

and unboundedness means

$$\forall\, t \in \mathbb{R}^+: \exists\, r \in \mathbb{R}^+: \Theta(r) > t$$

Each time transformation defines a mapping $\lambda\, s : \Theta \circ s$ on dense streams

$$s \in M^{\mathbb{R}}$$

by functional composition $\Theta \circ s$ which is defined by

$$(\Theta \circ s).t = s.\Theta(t)$$

By $\prod$ we denote the set of time transformations. A function F on continuous streams is called *time independent* if for every time transformation $\Theta \in \Pi$ we have

$$\{\Theta' \circ y: y \in F.x \wedge \Theta' \in \Pi\} = \{\Theta' \circ y: y \in F.(\Theta \circ x) \wedge \Theta' \in \Pi\}$$

Then the messages in the output streams F are independent of the timing of the input messages.

3.2 Composition Operators

When modelling systems and system components the composition of larger systems from smaller ones is a fundamental principle. We consider only one basic composition operator, namely parallel composition with feedback. It comprises *sequential* composition, *parallel* composition, and *feedback* as special cases. As is well known, these three composition operators suffice to formulate all kinds of finite networks of reactive information processing components.

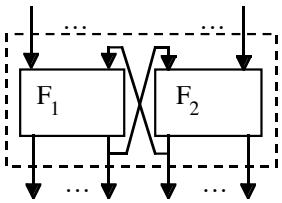

Fig. 2 Parallel Composition with Feedback

To define parallel composition with feedback we first introduce an operation that allows us to form a valuation (of channel by streams) out of two valuations (of channels of streams). Given two disjoint sets of channels C_1 and C_2 and two valuations

$$x \in \vec{C}_1,\ y \in \vec{C}_2$$

for them we construct a valuation

$$x \oplus y \in C_1 \cup \vec{C}_2$$

as follows:

$$(x \oplus y).c = x.c \qquad \text{if } x \in \vec{C}_1$$

$$(x \oplus y).c = y.c \qquad \text{if } c \in \vec{C}_2$$

We assume that the sets of channels C_1 and C_2 are disjoint.

Given channels sets O_1, I_1, O_2, I_2 and two behaviours (where $O_1 \cap O_2 = \emptyset$, $O_1 \cap I_1 = \emptyset$, $O_2 \cap I_2 = \emptyset$)

$$F_1 : \vec{I}_1 \to \wp(\vec{O}_1), \qquad F_2 : \vec{I}_2 \to \wp(\vec{O}_2)$$

we define the parallel composition with feedback by the function

$$F_1 \otimes F_2 : \vec{I} \to \wp(\vec{O})$$

where

$$I = (I_1 \backslash O_2) \cup (I_2 \backslash O_1),\ O = (O_1 \backslash I_2) \cup (O_2 \backslash I_1)$$

specified by (with the valuation $y \in \vec{C}$ where $C = O_1 \cup O_2 \cup I_1 \cup I_2$)

$$(F_1 \otimes F_2).x = \{y|O: \quad x|I = y|I \wedge$$
$$y|O_1 = F_1(y|I_1) \wedge$$
$$y|O_2 = F_2(y|I_2) \qquad \}$$

Here by $y|O$ we denote the restriction of the valuation mapping $y \in \vec{C}$ to the channel set $O \subseteq C$. Thus

$$y|O \subseteq \vec{O} \text{ and } (y|O).c = y.c \text{ for } c \in O.$$

Note that according to our semantic model the component specification $F_1 \otimes F_2$ is realisable provided the components F_1 and F_2 are realisable. This is a consequence of the following theorem (see [Müller, Scholz 97] for a proof on the basis of metric spaces).

Theorem: Every realisable time guarded behaviour F (with input and output channels C and delay) has a fixpoint.

Proof: Let f be a time guarded function such that f.x ∈ F.x for all x. We construct a sequence of valuations of the channels in C streams $s_i \in \bar{C}$, where s_0 is arbitrary and

$$s_{i+1}\downarrow(1+i)*\delta = f(s_i)\downarrow(1+i)*\delta$$

Such a sequence of streams exists and $s_i\downarrow i$ is uniquely defined by this due to the delay property. By induction on $i \in \mathbb{N}$ we prove

$$s_i\downarrow(i*\delta) = s_{i+1}\downarrow(i*\delta).$$

For i = 0 the equation is trivial. Assume the equation holds for i. Since

$$s_{i+1}\downarrow(i*\delta) = s_i\downarrow(i*\delta)$$

we have by the delay property:

$$f(s_{i+1})\downarrow((i+1)*\delta) = f(s_i)\downarrow((i+1)*\delta) \qquad (*).$$

We obtain

$$s_{i+2}\downarrow((i+1)*\delta) = \qquad \{\text{by def.}\}$$
$$f(s_{i+1})\downarrow((i+1)*\delta) = \qquad (*)$$
$$f(s_i)\downarrow((i+1)*\delta) = \qquad \{\text{by def.}\}$$
$$s_{i+1}\downarrow((i+1)*\delta)$$

This implies that $s_i\downarrow(i*\delta) = s_j\downarrow(i*\delta)$ for all j, $i \le j$. Therefore exists a uniquely defined stream s such that

$$s\downarrow(i*\delta) = s_i\downarrow(i*\delta)$$

for all $i \in \mathbb{N}$. By construction we have the equation

$$s = f(s). \qquad \square$$

It is not difficult to prove that the communication history s is the only fixpoint of f. If

$$r = f(r)$$

holds then by induction on i we can prove

$$r\downarrow(1*\delta) = s_i\downarrow(1*\delta)$$

for all i. Therefore r = s.

This theorem shows that the quite realistic assumption that for every component there exists a minimal non-zero reaction time has a pleasant effect: the existence of fixpoints is guaranteed and in the case of deterministic components the fixpoint is unique.

3.4 Recursion

Often we want to define behaviours and components by recursion. This is useful both for the description of infinite networks and for the recursive declaration of behaviours. To do this we use a function

$$\tau: (\vec{I} \to \wp(\vec{O})) \to (\vec{I} \to \wp(\vec{O}))$$

which transforms behaviours. We write

$$F_1 \subseteq F_2$$

if

$$\forall\, x \in \vec{I} : F_1(x) \subseteq F_2(x))$$

We call the function τ *inclusion monotonic*, if the following formula holds:

$$F_1 \subseteq F_2 \Rightarrow \tau[F_1] \subseteq \tau[F_2]$$

For an inclusion monotonic τ we choose a fixpoint $F : \vec{I} \to \wp(\vec{O})$ defined by the equation

$$F = \tau[F]$$

which is inclusion largest. A motivation for this choice is given below. We define the component behaviour function $F : \vec{I} \to \wp(\vec{O})$ by recursion on the function τ as follows. We define:

$$F = \text{fix}\ \tau$$

where

$$(\text{fix}\ \tau).x = \bigcup \{G.x: G \subseteq \tau[G]\}$$

According to this definition fix τ is the inclusion largest set-valued function F such that the equation

$$F = \tau[F]$$

holds. F is not the only fixpoint, in general. Besides the inclusion largest fixpoint there may be many others.

Recursion can be used for two purposes. We may use recursion when writing specifications where the behaviour transformer τ is specified by a logical expression defining the set $\tau[F].x$ in terms of F. We may define $\tau[F]$ also by a number of operators for forming a network from F and a number of given components by parallel composition. Then fix τ can be interpreted to represent a recursively defined infinite network.

As well-known, the difficulty of associating a unique fixpoint with a fixpoint equation are caused by unguarded recursion. If a recursive call occurs without any modifications we obtain an infinite regression (also called divergence) which does not define the result of the recursive call uniquely. Therefore the fixpoint is not unique. This is different, if the recursion is guarded, in our case time guarded. Time guardedness is based on the idea that each call requires some delay in time. Unbounded regression requires an unbounded amount of time corresponding to empty result streams over an infinite time internal. We speak of a *time contracting* function τ.

We call a behaviour transformation τ *time contracting*, if for all behaviours F we have

$$TG(F, i) \Rightarrow TG(\tau[F], i+1)$$

Here TG(F, i) expresses that F is *time guarded by i time ticks*, formally (for simplicity we work with discrete time here and with time ticks representing a delay by one time unit here; we could also work with arbitrary time delays $\delta > 0$)

$$TG(F, i) \equiv \forall\, j \in \mathbb{N}: x{\downarrow}j = z{\downarrow}j \Rightarrow F(x){\downarrow}i+j = F(z){\downarrow}i+j$$

If τ is time contracting in the sense defined above, then the safety part of fix τ is unique. The safety part is determined by the prefix closure close(F.x) of the set F.x. It is formally determined by (let $W \subseteq \vec{C}$)

$$close(W) = \{x{\downarrow}n: n \in \mathbb{N}^+ \wedge x \in W\}$$

The uniqueness of the safety part of the fixpoint is shown by the following theorem.

Theorem: Let τ be inclusion monotonic and time contracting. Then for all realisable functions F, F' with $F = \tau[F]$ and $F' = \tau[F']$ their safety parts coincide. Mathematically speaking:

$$\forall\, i \in \mathbb{N}: F{\downarrow}i = F'{\downarrow}i$$

Proof: Define functions $F_i \in \vec{I} \rightarrow \wp(\vec{O})$ by (let F_0 be arbitrary, but realisable) such that

$$(F_{i+1}){\downarrow}i+1 = \tau[F_i]{\downarrow}i+1$$

By the construction and the contractivity of τ we have for all time points $i \in \mathbb{N}$ and all fixpoints $F = \tau[F]$:

$$F{\downarrow}i = (F_i){\downarrow}i = F'{\downarrow}i \qquad \square$$

If $\tau[F]$ is formed by applying the parallel composition to F and a number of basic components, then τ is obviously inclusion monotonic. It is time contracting, if F is sequentially composed with a time guarded function. A simple way to achieve this is to take a one time unit delay function Δ that puts a time tick in front of each of its input streams. In mathematical terms the delay function (for an arbitrary channel set C)

$$\Delta: \vec{C} \rightarrow \vec{C}$$

is defined by the equation (for $x \in \vec{C}$, $c \in C$, $t \in \mathbb{N}^+$):

$$((\Delta.x).c).t = (x.c).(t+1)$$

We replace $\tau[F]$ by $\tau[F ; \Delta]$. This construction corresponds to the simple assumption that a recursive call takes exactly one unit of time.

By $F_1 ; F_2$ denote the sequential composition of the two components F_1 and F_2 defined by

$$(F_1 ; F_2).x = \{z \in F_2(y): y \in F_1(x)\}.$$

Let us explain this construction by a simple example. If we choose the trivial example of recursion where the behaviour transformation τ is the identity function where formally we always have:

$$\tau[F] = F$$

then the replacement leads to the fixpoint equation:

$$F = F ; \Delta$$

The only solution of this equation in the set of time guarded functions is the function that produces only time ticks (or in our presentation produces an infinite stream of empty sequences).

We need more sophisticated arguments to justify our choice to associate the inclusion largest fixpoint with τ. In the case of a non-contractive function τ "divergence" is simply represented by chaos. In the case of time contractive functions, divergence is modelled by the output stream carrying only time ticks. This is exactly what we intend to associate with a diverging computation. No output is produced in finite time, but time does proceed. The construction is given for discrete streams but can be carried over to dense streams. There, however, we have to find a representation of the fact that there is no message on the stream for a time unit δ. This can easily be expressed by a dummy message.

4. Refinement

Refinement is the basic concept for the development of components. We describe two basic forms of refinement: *property refinement* and *interaction refinement*. Property refinement allows us to replace a behaviour by one with additional properties, in other words, by one fulfilling more requirements.

4.1 General Concepts of Refinement

A behaviour

$$F_1: \vec{I} \rightarrow \wp(\vec{O})$$

is refined by a behaviour

$$F_2: \vec{I} \rightarrow \wp(\vec{O})$$

if

$$F_2 \subseteq F_1$$

Property refinement does not allow us to change the syntactic interface of a component. This can be done by interaction refinement, however.

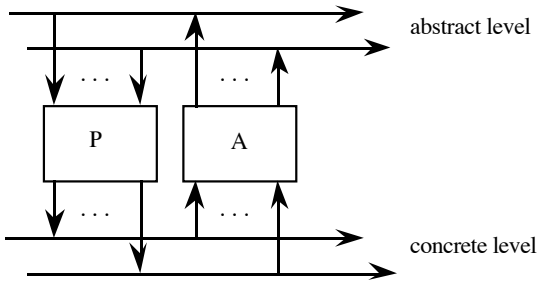

Fig. 3 Communication History Refinement

By interaction refinement we may change the number of input and output channels of a system but still relate the behaviours in a formal way. A *communication history refinement* requires timed functions

$A: \vec{I} \to \wp(\vec{O}), \qquad P: \vec{O} \to \wp(\vec{I})$

where

$P ; A = Id$

Let Id denote the identity relation

$Id.x = \{x\}$

Fig. 3 gives the "commuting diagram" of history refinement.

Based on the idea of a history refinement we introduce the idea of an interaction refinement for components.

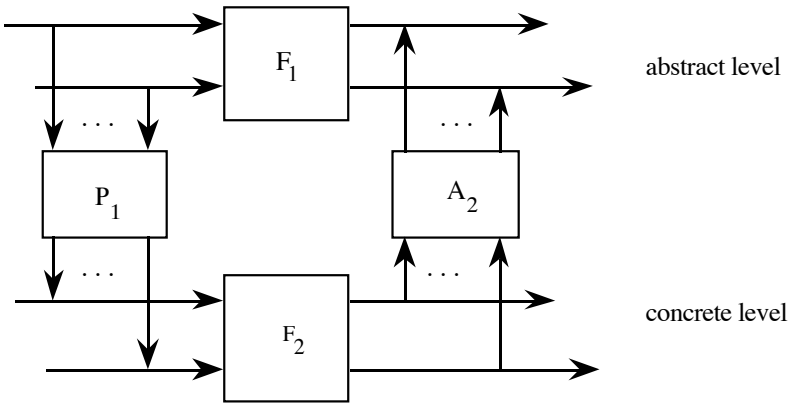

Fig. 4 Commuting Diagram of Interaction Refinement (*U-simulation*)

Given two communication history refinements

$A_1: \vec{I_1} \to \wp(\vec{I_2}) \qquad P_1: \vec{I_2} \to \wp(\vec{I_1})$

$A_2: \vec{O_1} \to \wp(\vec{O_2}) \qquad P_2: \vec{O_2} \to \wp(\vec{O_1})$

we call the behaviour

$F_2: \vec{I_2} \to \wp(\vec{O_2})$

an *interaction refinement* of the behaviour

$F_1: \vec{I_1} \to \wp(\vec{O_1})$

if one of the following four proposition holds

$P_1 ; F_2 ; A_2 \subseteq F_1$	*U-simulation*
$P_1 ; F_2 \subseteq F_1 ; P_2$	*downward simulation*
$F_2 ; A_2 \subseteq A_1 ; F_1$	*upward simulation*
$F_2 \subseteq A_1 ; F_1 ; P_2$	*U^{-1}-simulation*

Note that U^{-1}-simulation is the strongest condition from which all others follow. Note, moreover, that property refinement is a special case of interaction refinement (choose for A_i and P_i the identity relation).

4.2 Refinement of Time

In this section we study interaction refinements that allow us to do refinements from non-timed streams to discrete time and further on to continuous time and finally to dense streams.

4.2.1 From Non-timed to Timed Streams

A non-timed stream can be seen as the abstraction from all discrete timed streams with the same message set but arbitrary timing. The abstraction function A is simple:

$$A.y = \{ \overline{y} \}$$

The representation function is also simple:

$$P.x = \{y: \overline{y} = x\}.$$

Although the abstraction is so simple, forgetting about time has serious consequences for functions that describe the behaviour of components. Time guarded gets lost and as a consequence also the uniqueness of fixpoints which has crucial impacts on the compositionality of the semantic models (for an extensive discussion, see [Broy 93]).

4.2.2 From Discrete to Continuous Time

Given a discrete, equidistant timed, time discrete stream $s \in M^{\aleph}$ we get a abstraction function

$$\alpha.\ M^{\Re} \to M^{\aleph}$$

It is specified as follows

$$(\alpha.r).j = r.j$$

$$(\alpha.r)^{\tau} j = \min \{n \in \mathbb{N}: r^{\tau} j \leq n\}$$

We define the abstraction relation

$$A: M^{\Re} \to \wp(M^{\aleph})$$

by

$$A.r = \{ \alpha.r \}$$

and the representation specification:

$$R: M^{\aleph} \to \wp(M^{\Re})$$

by

$$R.s = \{r: s = \alpha.r \}$$

The step from discrete to continuous time (or back) is rather simple and does not have much consequences for the semantic techniques, as long as the time granularity is chosen fine enough to keep time guardedness.

4.2.3 From Discrete Streams to Dense Streams

To abstract a dense stream into a discrete stream we can use the following two techniques:

- sampling,
- event discretisation.

In sampling we select a countable number of time points as samples. We define the abstraction specification

$$A: M^{\mathbb{R}} \to \wp(M^{\mathfrak{N}})$$

that maps dense to discrete streams by

$$A.r = \{ \alpha.r \}$$

where (we choose a simple variant of sampling where we select the natural numbers as the sample time points)

$$(\alpha.r).i = r.i$$

$$(\alpha.r)^{\tau} j = j$$

A representation specification for *sampling* in continuous time is obtained by the function

$$R: M^{\mathfrak{N}} \to \wp(M^{\mathbb{R}})$$

defined by (we ignore for simplicity the possibility of successive identical messages)

$$R.s = \{r: s = \alpha.r \}$$

Besides sampling the step from dense streams to discrete streams can be defined by *event discretisation*. An event in a dense stream is, for instance, that the values in a real numbered stream reach a particular given number or that the values do not increase any more. This can be modelled by a predicate

$$e: ([t: t+\delta] \to M) \to \mathbb{B}$$

which indicates by e(z) = true that for the behaviour z in the time interval [t: t+δ] the event characterised by e has occurred. We assume that the event characterised by e is only considered to occur once in a distance δ. For instance, we may consider the time interval in continuous time that a time point i represents in discrete time and define that an event occurs at time point i in discrete time if it occurs at least once in the respective interval. With the help of such predicates we may abstract a dense stream into a stream of sets of events. Note that sampling can be seen as a special case of modelling dense streams by discrete streams of events.

Using these concepts the specification of a sensor can be described. A sensor is a component with no input but a timed stream as output. We assume that an analog/digital-unit is inside the sensor that sends a signal in a predefined frequency.

An example of analog/digital converter that converts a dense stream of real numbers into a discrete stream of events as described above or by sampling.

There are many ways to obtain time abstractions. Typical examples are abstractions

- from dense streams to discrete streams with continuous time,
- from discrete streams with continuous to discrete time,
- from a finer discrete time to a coarser discrete time,
- from timed to non-timed streams.

In fact, all three steps mean that we use a coarser time model. This way we lose information about the timing of messages. As a consequence, messages at different time points may be represented by identical time points. This means we may lose the idea of causality. This leads to intricate problems as we find them in the approaches that work with the assumption of so-called *perfect synchrony*.

4.3 Time Synchrony versus Time Asynchrony

We call a system model *time synchronous* if a component may react to input instantaneously at the same point of time. Physically this seems to be impossible - we may assume that reaction always takes time. However, as an abstraction from a behaviour this might be reasonable. With this concept of time synchrony we lose the property of time guardedness. As a consequence, fixpoints of feedback loops are no longer unique, and in the case of pathological programs may not even exist.

We study the transition from an asynchronous model of time to a model with perfect synchrony on the level of discrete time. To demonstrate the problems we use a simple delay function

$$\Delta : M^{\aleph} \rightarrow \wp(M^{\aleph})$$

defined by (for $x \in M^{\aleph}$)

$$(\Delta.x):(i+1) = \{x:i\}$$
$$(\Delta.x):0 = \{\langle\rangle\}$$

The abstraction that maps a finer time granularity to a coarser time granularity by combining a number of time interval is given by the function

$$A : M^{\aleph} \rightarrow \wp(M^{\aleph})$$

defined by

$$(A.x):i = x:(i*k)^\wedge(x:(i*k+1))^\wedge ... \ ^\wedge(x:((i+1)*k-1))$$

the representation specification

$$R : M^{\aleph} \rightarrow \wp(M^{\aleph})$$

is simply defined by the requirement

$$(R; A).x = \{x\}$$

Informally speaking R maps each sequence of messages in a time interval in the stream x onto k intervals of k sequences of message such that the concatenation of these messages yields the interval again.

If we apply this refinement pair to Δ we get (by U^{-1}-simulation) the requirement for the refinement Δ'.

$$\Delta' = A; \Delta; R$$

In contrast to Δ the refinement Δ' cannot be time guarded (for $k > 1$). A fixpoint of Δ is always the empty stream since we obtain from

$$x \in \Delta.x$$

the equation

$$x{:}0 = \langle\rangle$$

$$x{:}(i+1) = x{:}i$$

For Δ' we obtain the fixpoint equations

$$(x{:}0)^\frown \ldots {}^\frown(x{:}k-1) = \langle\rangle^\frown \ldots {}^\frown x{:}k-2$$

$$(x{:}i*k)^\frown \ldots {}^\frown(x{:}(i+1)*k-1) = (x{:}(i*k)-1)^\frown \ldots {}^\frown(x{:}(i+1)*k-2)$$

for which we can find many streams that fulfil these equations (a simple solution is every stream with

$$x{:}((i+1)*k-1) = \langle\rangle.$$

This simple analysis shows that we lose the simplicity of fixpoint theory for time guarded behaviours since perfect synchrony leads to behaviours that are not time guarded and therefore a much more sophisticated fixpoint theory is required.

5. Conclusions

The specification of interactive systems has to be done in a time/space frame. A specification should indicate which events (communication actions) can take place where, and when, and how they are causally related. Such specification techniques are an important prerequisite for the development of safety critical systems.

Modelling information processing systems appropriately is a matter of choosing the adequate abstractions in terms of the corresponding mathematical models. Giving operational models that contain all the technical computational details of interactive nondeterministic computations is relatively simple. However, for control system engineering purposes operational models are not very helpful. Only, if we manage to find good abstractions is it possible reach a tractable basis for system specifications.

Abstraction means forgetting information. Of course, we may forget only information that is not needed. Which information is needed does not only depend upon the explicit concept of observation, but also upon the considered forms of the composition of systems from subsystems.

Finding appropriate abstractions for operational models of distributed systems is a difficult but nevertheless important task. Good abstract non-operational models are the basis of a discipline of system development.

Time information can be treated as any other information, except, however, that the time flow follows certain laws. This is expressed by the timing requirements.

Acknowledgement

I am grateful to Ketil Stølen for a number of discussions that were helpful to clarify the basic concepts. It is a pleasure to thank my colleagues Olaf Müller and Jan Philipps for many helpful remarks on a draft version of this paper.

Appendix A: A Short Note on Time Guardedness

Trivially $x\downarrow 0$ is the empty sequence. Let

$$F: M^{\mathbb{R}} \rightarrow \wp(M^{\mathbb{R}})$$

be a time guarded function with time delay δ. From time guardedness we can conclude that either $F.x = \emptyset$ for all streams x or for none of the streams x. Since by the definition of time guardedness

$$x_1\downarrow 0 = x_2\downarrow 0 \qquad \text{for all input histories } x_1, x_2$$

and therefore

$$F.x_1\downarrow \delta = F.x_2\downarrow \delta \qquad \text{for all } x_1 \text{ and } x_2$$

From this we may conclude that if $F.x_1 = \emptyset$ for any x_1 we have $F.x_2 = \emptyset$ for all x_2.

We can define time guardedness also in a more liberal style that does not exclude partially inconsistent specifications. If we have a partially inconsistent function F we define

$$\text{dom } F = \{x: F.x = \emptyset\}$$

Then we define time guardedness by

$$\forall\, x_1, x_2 \in \text{dom } F: x_1\downarrow i{*}\delta = x_2\downarrow i{*}\delta \Rightarrow F.x_1\downarrow (i+1){*}\delta = F.x_2\downarrow (i+1){*}\delta$$

Note that we cannot simply turn a partially inconsistent specification into a consistent specification by adding chaos since the result will not be time guarded, in general. We define the function F^a by:

$$F^a.x = \textbf{if } x \in \text{dom } F \textbf{ then } F.x \textbf{ else } M^{\mathbb{R}} \textbf{ fi}$$

F^a is **not** time guarded in general. We may define instead

$$F^c = \cup \{Q: Q \text{ is time guarded} \wedge \forall\, x \in \text{dom } F: Q.x = F.x\}$$

F^c is inclusion the largest time guarded relation that coincides with F on dom F. It can be constructed by defining for every $x \notin \text{dom } F$ a sequence of sets Y_i where Y_i is defined by

$$Y_i = \cup \{y: \forall\, z: z\downarrow i{*}\delta = x\downarrow i{*}\delta \wedge z \in \text{dom } F \Rightarrow y\downarrow (i+1){*}\delta = F.z\downarrow (i+1){*}\delta\}$$

We define

$$F^c.x = \cap\, Y_i$$

References

[Alur, Dill 94]
R. Alur, D. Dill: A theory of timed automata. Theoretical Computer Science 126, 1994, 183-235

[Alur et al. 95]
R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine: Agorithmic analysis of hybrid systems. Theoretical Computer Science 138, 1995, 3-34

[Baeten, Bergstra 91]
J.C.M. Baeten, J.A. Bergstra: Real Time Process Algebra. Formal Aspects of Computing 3, 1991, 142-188

[Broy 83]
M. Broy: Applicative real time programming. In: Information Processing 83, IFIP World Congress, Paris 1983, North Holland Publ. Company 1983, 259-264

[Broy 93]
M. Broy: Functional Specification of Time Sensitive Communicating Systems. ACM Transactions on Software Engineering and Methodology 2:1, Januar 1993, 1-46

[Broy, Stølen 94]
M. Broy, K. Stølen: Specification and Refinement of Finite Dataflow Networks – a Relational Approach. In: Langmaack, H. and de Roever, W.-P. and Vytopil, J. (eds): Proc. FTRTFT'94, Lecture Notes in Computer Science 863, 1994, 247-267

[Lynch, Stark 89]
N. Lynch, E. Stark: A proof of the Kahn principle for input/output automata. Information and Computation 82, 1989, 81-92

[Lynch, Tuttle 87]
N. A. Lynch, M. R. Tuttle: Hierarchical correctness proofs for distributed algorithms. In: Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing, 1987

[Lynch, Vaandrager 96a]
N. Lynch, F. Vaandrager: Action Transducers and Time Automata. Formal Aspects of Computing 8, 1996, 499-538

[Lynch, Vaandrager 96b]
N. Lynch, F. Vaandrager: Forward and Backward Simulations, Part II: Timing-Based Systems. Information and Computation 128:1, 1996

[Müller, Scholz 97]
O. Müller, P. Scholz: Functional Specification of Real-Time and Hybrid Systems. In HART'97, Proc. 1st Int. Workshop on Hybrid and Real-Time Systems, to appear in LNCS, 1997

[Park 80]
D. Park: On the semantics of fair parallelism. In: D. Björner (ed.): Abstract Software Specification. Lecture Notes in Computer Science 86, Berlin-Heidelberg-New York: Springer 1980, 504-526

[Park 83]
D. Park: The "Fairness" Problem and Nondeterministic Computing Networks. Proc. 4th Foundations of Computer Science, Mathematical Centre Tracts 159, Mathematisch Centrum Amsterdam, (1983) 133-161

[Pannangaden, Shanbhogue91]
P. Pannangaden, V. Shanbhogue: The expressive power of indeterminate dataflow primitives. Information and Computation 98, 1992, 99-131