T U M

I N S T I T U T F Ü R I N F O R M A T I K

Formal Semantics of Time Sequence Diagrams

Christian Facchi

TUM-I9540
Dezember 1995

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-12-1995-I9540-300/1.-FI

Typescript: ---

Druck: Mathematisches Institut und
Institut für Informatik der
Technischen Universität München

Formal Semantics of Time Sequence Diagrams

Christian Facchi
Institut für Informatik
Technische Universität München
D-80290 München
E-Mail: facchi@informatik.tu-muenchen.de

December 29, 1995

Abstract

Time Sequence Diagrams (TSDs) are a graphical representation employed to clarify the communication between service users and a service provider in the ISO/OSI basic reference model. In this paper we define the syntax and semantics of a textual representation for TSDs. As well, we provide a method for translating TSDs into this language. Furthermore, some extensions of TSDs are introduced that allow some special facets of an arbitrary layer to be described.

1 Introduction

The ISO/OSI basic reference model [ISO84a, CCI88a] describes different layers for the modularization of computer communication. According to [VL86] a specification of one layer should start at the highest level of abstraction. Therefore, service specifications are need as requirements rather than protocol specifications, which can only describe the implementation of a service. Thus, every layer has an ISO or CCITT[1] document which specifies its services. These specifications are in natural language, supplemented with some additional formal and semi-formal material. The natural language and the semi-formal parts may lead to unintended interpretation ambiguities, which should be avoided in an international standard. To avoid misinterpretation we believe that it is necessary to develop a formal service specification for every layer. Therefore, we propose that the semi-formal parts of such specifications should be transformed into formal ones in a schematic way. Toward this goal, we provide such a transformation for Time Sequence Diagrams. (Translations for other semi-formal parts can be found in [Fac95].) Later the natural language part of such specifications, which is usually layer specific, can be formally specified. The different specification parts are then combined into a complete service specification. In [VSvSB91] this is called a constraint oriented specification style. With this approach a pure formal specification of a service can be derived.

We note that the method presented in this paper only provides a proposed formal semantics for TSDs and has not been authorized by a standardization organization.

[1] In 1993 the CCITT became the Telecommunication Standards Sector of the International Telecommunication Union (ITU-T). If a document is published by CCITT, this organization name is used instead of ITU-T in the sequel.

Our method is only based on examples given in the ISO/CCITT documents or in literature.

Section 2 introduces the basic concepts and formalisms. In Section 3 the syntax of a simple language (TSD-DL) for the representation of TSDs is introduced. For this language a denotational semantics, which is based on traces, is given in Section 4. A development of TSD-DL expressions for arbitrary TSDs is the main part of Section 5. In Section 6 some example TSD-DL expressions for TSDs are presented. In Section 7 extensions of TSDs are proposed and discussed. Section 8 presents a semantics without restriction of the service user's behavior.

2 Basic Concepts

In this section we give a short introduction to Time Sequence Diagrams and our formalism.

2.1 Time Sequence Diagrams

Time sequence diagrams, in the sequel abbreviated as TSDs, are a semi-formal means of describing a specific property of a service specification. TSDs are defined in [ISO87a, CCI88b] (a newer version is in [ISO94]) by some examples. We restrict our semantic foundation to the first definitions, because the basic reference model [ISO84a] is based on them. In the sequel we will discuss the differences between the two definitions of TSDs.

TSDs are considered to be at most semi-formal, e.g. in [BHS91], because they are described in [ISO87a, CCI88b] without a formal semantics. It is our intention to show that and how they can be formalized.

With TSDs timing precedences between service primitives can be expressed. In the following, the term "*events*" is used instead of "service primitives" as an abbreviation.

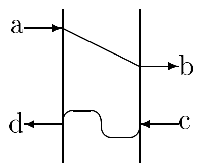

Figure 1: Example TSD

In Figure 1, a TSD describing the temporal ordering of the events a, b, c and d is presented. The two vertical lines, which denote the Service Access Points (SAP), divide three different communication partners from left to right: User A (on the left-hand side), the service provider (between the two vertical lines) and user B (on the right-hand side). The vertical lines also denote an increase in time downwards. Therefore a temporal ordering of service primitives at each SAP is given. Two service primitives at different SAPs can be related by a solid line, which indicates a timing precedence. For example in Figure 1 the event b takes place after event a.

Events connected with a tilde ($\sim$) are not related with respect to time. Therefore any temporal ordering of the events c and d in Figure 1 is allowed.

A service specification uses a set of TSDs to describe service behavior. Therefore it is necessary to define a combination of TSDs. However, the interpretation of the combination is hidden in the graphical representation of TSDs and therefore has to be defined in the semantics of TSDs. In the following we will informally introduce our interpretation of the combination of TSDs. Therefore, we define some notions in an informal way which will later be formalized.

The TSDs of a service specification can be regarded as a description of the generation principle for the allowed sequences of events, which we call *scenarios*. To clarify the semantics of a TSD description of a service we introduce the notion of *scenaric interpretation*, which describes one possible scenario of a TSD without repetition. The *complete interpretation* describes an arbitrary number of repetitions of TSDs. The scenaric interpretation of a single TSD describes the allowed scenarios of events, where it is not possible that only a prefix of these scenarios take place. In other words the scenaric interpretation requires that all events of a TSD take place.

The alternative part of the combination of TSDs is defined by an arbitrary choice of the scenarios described by single TSDs. The complete interpretation of TSDs covers the repetitive part of the combination which is defined by an arbitrary interleaving of one scenario with the repetition of this scenario. The complete interpretation of a TSD is derived by an interleaving of an arbitrary number of scenarios which are the result of a scenaric interpretation. This view is necessary to describe a possibly reordering of data during the data transfer in lower layers. A detailed motivation of the chosen combination is given in [Fac95].

TSDs describe neither a safety nor a liveness property in the sense of [AS87]. They express a combined safety and liveness property. This is a consequence that due to the chosen scenaric interpretation of TSDs all events have to happen.

2.2 Traces

With a global view of time the timing precedence of two events at different SAPs can be examined. Therefore traces are chosen as a semantic basis. Such a set of traces can be used for the interpretation of TSDs.

Traces are used to describe a history of events in distributed systems. They are possibly infinite sequences of events, which are denoted by the type $Trace\ Event$ [2].

- $\&$ is used as constructor and ϵ as empty trace.$<a_1, a_2, \ldots, a_n>$ is an abbreviation for $a_1 \& a_2 \ldots \& a_n \& \epsilon$.
- $\circ$ denotes the concatenation. If s is an infinite trace $s \circ t = s$ holds.
- $\sqsubseteq$ denotes the usual prefix ordering: $s \sqsubseteq t = \exists s'.\ (s \circ s' = t)$
- The filter function $a \copyright t$ yields the subpart of the trace t consisting only of the elements a. E.g. the following holds: $a \copyright <a, c, a, b, a> = <a, a, a>$ For simplicity a set can be used as first operand: $\{a, c\} \copyright <a, c, a, b, a> = <a, c, a, a>$

[2]Note that $Trace$ is used as a type constructor. $Trace\ Event$ is an instantiation of the polymorphic type $Trace\ \alpha$

- $\#$ yields the length of a trace. The length of an infinite trace is ∞.
- fix denotes the fixpoint operator, which yields the least fixpoint of a function. E.g. $\mathsf{fix}\boldsymbol{\lambda} s.1\&s$ denotes an infinite trace consisting of only 1's. For a detailed explanation see e.g. [LS87].

As usual with traces an interleaving semantics is taken in the following in which simultaneous happening events can not be expressed. This can be extended to real concurrency, in which simultaneous events are possible, by introduction of a set of events instead of events as basic elements of traces. This would make the notation more complicated.

A more detailed introduction to the formalism we use can be found in [BDD+93].

For the definition of the formal semantics of TSDs we introduce the strict function *filter*. The filter function is polymorphic, because it is later used with traces of different sorts.

$$\text{filter: } \alpha \times \text{Trace } \alpha \times \text{Trace } \beta \longrightarrow \text{Trace } \beta$$

$$\text{ps} = \epsilon \vee \text{xs} = \epsilon \Rightarrow \text{filter(el,ps,xs)}=\epsilon$$

$$\begin{aligned}\text{filter(el,p\&ps,x\&xs)} = \mathbf{if}\ & (\text{el} = \text{p}) \\ \mathbf{then}\ & \text{x\&filter(el,ps,xs)} \\ \mathbf{else}\ & \quad \text{filter(el,ps,xs)}\end{aligned}$$

The function $filter(el, ps, xs)$ extracts the trace of all elements of a trace xs whose elements at the corresponding position in ps are equal to the element el. ps is later used as a prophecy parameter to simulate an arbitrary segmentation of a trace.

Example 1: The $filter$ Function

In this example the effect of the $filter$ function is demonstrated.

$$filter(1, <1, 0, 1, 2>, <a, b, c, d>) = <a, c>$$

$$filter(4, <1, 0, 1, 2>, <a, b, c, d>) = <>$$

Even infinite traces can be used:

$$filter(1, <1, 1, 2, 2, \ldots>, <a, b, a, b, \ldots>) = <a, b>$$

$$filter(1, <1, 2, 1, 3, 4, 2, \ldots>, <a, a, b, a, a, b, \ldots>) = <a, b>$$

$\square$

3 Syntax of TSD-DL

In this section the BNF syntax of the Time Sequence Diagram Description Language (TSD-DL) is defined. The basic elements of this language are *events* and *operations*. An event is a service primitive with additional information about the direction and the source. Initially, we omit this additional information.

(1) <TSD> ::= <Event>
(2) <TSD> ::= <TSD> $\rightarrow$ <TSD>
(3) <TSD> ::= <TSD> $\sim$ <TSD>
(4) <TSD> ::= <TSD> $\circledast$ <TSD>
(5) <TSD> ::= <TSD> | <TSD>
(6) <TSD> ::= <TSD>$\overset{\omega}{-}$

In line 1-4 the elements for the construction of a single TSD are introduced. Line 5 and 6 represent combination operators which we use to express how TSDs are put together. Two elements can be sequentially combined by the operator $\rightarrow$. The absence of a timing relation between two elements is described by the $\sim$ operator. The conjunctive combination of timing precedences is described by $\circledast$. The alternative composition of separated TSDs is achieved by the $|$ operator. The last combination operator is $\overset{\omega}{-}$, that describes a repetition of a TSD-DL expression.

In the following, we give a few examples of how TSDs can be expressed by TSD-DL.

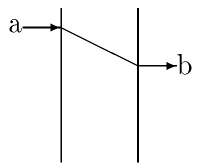

Figure 2: TSD with two elements

The TSD shown in Figure 2 defines that the event a leads later to an event b. This timing precedence, which describes the scenaric interpretation, can be expressed by the TSD-DL expression $a \rightarrow b$. Note that both events have to take place. The $\rightarrow$ symbol denotes an additional timing precedence.

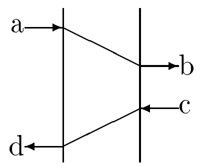

Figure 3: TSD with four elements

The TSD of Figure 3 describes a scenario with four events, where first a, then b, then c and finally event d takes place. This leads to the scenaric interpretation expressed with the TSD-DL expression $(a \rightarrow b) \rightarrow (c \rightarrow d)$. Because $\rightarrow$ is associative also $a \rightarrow (b \rightarrow (c \rightarrow d))$ can be used, but the first expression is closer to the graphical representation.

A description of all TSDs of a service is: $(\mathrm{TSD}_1 \mid \mathrm{TSD}_2 \mid \ldots \mid \mathrm{TSD}_n)\overset{\omega}{-}$. Here TSD_i is an arbitrary TSD-DL expression for the scenaric interpretation of a single

TSD. The intuition behind this TSD-DL expression is that each TSD_i describes the scenaric interpretation of one TSD. All TSDs of one layer can be alternatively combined using $|$. This leads to the scenaric interpretation of all TSDs of one layer: $(\text{TSD}_1 \mid \text{TSD}_2 \mid \ldots \mid \text{TSD}_n)$. However, the scenaric interpretation describes exactly possible sequences of events of TSDs without repetition. An arbitrary number of repetitions, in other words the complete interpretation, is achieved by the operator $\underline{\omega}$.

4 A Denotational Semantics of TSD-DL Expressions

A TSD-DL expression describes a set of traces of events. To simplify the description, predicates are used to denote sets. In the following a TSD-DL expression TS can be inductively transformed into a predicate $P_{TS}(t)$ with the following intuition:

> $P_{TS}(t)$ holds iff a trace t of events, which is denoted by the sort $Trace\ Event$, is correct with respect to the construction rules described in the TSD-DL expression TS.

The denotational semantics of a TSD-DL expression is defined as follows:

1. Basic TSDs:

$$P_a(\text{t}) = (\text{t}=\langle\text{a}\rangle) \qquad\qquad \text{iff a: Event}$$

Note that $P_a(t)$ holds only if the trace t consists of one element a.

2. Sequential Composition:

$$\begin{aligned} P_{TS_1 \rightarrow TS_2}(t) = \exists s_1, s_2 :& Trace\ Event. \\ & t = s_1 \circ s_2 \wedge P_{TS_1}(s_1) \wedge P_{TS_2}(s_2) \end{aligned}$$

3. Interleaving Composition:

$$\begin{aligned} P_{TS_1 \sim TS_2}(t) = \exists bs :& Trace\ Bool. \\ & \#bs = \infty \wedge P_{TS_1}(filter(true, bs, t)) \\ & \wedge P_{TS_2}(filter(false, bs, t)) \end{aligned}$$

4. Conjunctive Composition:

$$P_{TS_1 \circledast TS_2}(t) = P_{TS_1}(t) \wedge P_{TS_2}(t)$$

5. Disjunctive Composition:

$$P_{TS_1 | TS_2}(t) = P_{TS_1}(t) \vee P_{TS_2}(t)$$

6. Repetition:

$$P_{TS^{\underline{\omega}}}(t) = \exists ns : Trace\ Nat. \\ \#ns = \infty \wedge \forall n : \ nat.\ P_{TS}(filter(n, ns, t)) \\ \vee filter(n, ns, t) = \epsilon$$

The operator $\underline{\omega}$ is used to describe an arbitrary, possibly infinite fair interleaving of TSDs, i. e., a total segmentation of a trace into disjoint parts. E.g: $P_{(a \to b)^{\underline{\omega}}}(t)$ holds for the infinite trace $t = \mathsf{fix}\boldsymbol{\lambda} s.<a, b>\circ s$, because $ns = <1, 1, 2, 2, \ldots>$. Another example: $P_{(a \to b)^{\underline{\omega}}}(t)$ also holds for $t = \mathsf{fix}\boldsymbol{\lambda} s.<a, a, b>\circ s$, because $ns = <1, 2, 1, 3, 4, 2, \ldots>$. The last trace describes a property, which can not be approximated from finite behavior. For every finite prefix t' of t, $P_{(a \to b)^{\underline{\omega}}}(t')$ does not hold. Only in the infinite trace t, does there exist, for every event a, a corresponding event b.

Remark: Modification of the Semantics of Basic TSDs

If the empty trace would be included in the definition of basic TSDs in line 1, the basic property of TSDs that every event of a TSD has to take place is violated. This can be seen if the definition of the basic case $P_a(t) = (t = <a>)$ would be modified. In the following example the effects of including the empty trace in the definition of the semantics of basic TSDs is shown.

1'. Basic TSDs:

$$P'_a(\mathrm{t}) = (\mathrm{t}{=}{<}\mathrm{a}{>} \vee \mathrm{t}{=}\epsilon) \qquad \text{iff a: event}$$

The TSD-DL expression of the TSD shown in Figure 2 would have after a simplification the semantics using the same definition of the sequential composition given by line 2:

$$P'_{a \to b}(t) = (t = \epsilon \vee t = <a> \vee t = <b> \vee t = <a, b>)$$

This would give a trace which consists only of the event a a correct semantics. That is not intended by the TSD. □

The *semantics of a TSD-DL expression expr* is defined as the set of all traces where P_{expr} holds:

$$[\![expr]\!]_{\mathcal{TSD}} := \{t \in Act^\omega | P_{expr}(t)\}$$

In this, Act^ω is the set of all traces over the actions denoted by ACT.

To take stock, we have now defined a formal semantics of TSD-DL expressions. What remains is the transformation of the graphical representation of TSDs to TSD-DL expressions. Then we will have given the graphical representation of TSDs a formal semantics, using TSD-DL as an intermediate step.

5 Transformation of Time Sequence Diagrams to TSD-DL Expressions

In this section a formal interpretation of TSD into TSD-DL expressions is presented. We note that, because only an informal semantics of TSDs exists the presented formal semantics can only be validated and not formally compared.

5.1 A First Step to the Interpretation of TSDs

Every TSD is translated into a TSD-DL expression. In the graphical representation of a TSD two corresponding service primitives on different SAPs have a timing precedence relation, which is represented by $\rightarrow$, or no timing relation, which is represented by $\sim$. The combination of more TSD parts can be achieved by the temporal ordering on the time axis (see Figure 3).

The TSD of Figure 1 is used in [CCI88b, ISO87a] to explain the meaning of TSDs. Even this TSD leads to an interpretation difficulty. One possible interpretation of the TSD shown in Figure 1 is the TSD-DL expression: $(a \rightarrow b) \rightarrow (c \sim d)$. In this interpretation it is guaranteed that event d takes place after event b. A more rigorous approach is that $\sim$ describes a real absence of a timing relation. Then it can only be stated that event a takes place before event b, a before d and b before c. In this interpretation it is allowed, that event d takes place before event b. This leads to the TSD-DL expression $a \rightarrow ((b \rightarrow c) \sim d)$. Note that the second interpretation includes the first one and is therefore more general. The second interpretation is given in the service specification of the network layer in [ISO87b, CCI88d] as an extra verbal addition. In [CCI88c] the same TSD is used for the service specification of the data link layer, but no additional text about the interpretation is explicitly given.

The same ambiguity can be demonstrated by Figure 4.

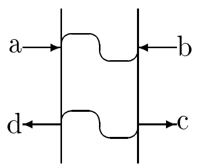

Figure 4: Example TSD

One interpretation of the TSD presented in Figure 4 is the TSD-DL expression $(a \sim b) \rightarrow (c \sim d)$. In this interpretation the events a and b have to take place before the events c and d. The second possible interpretation is that there are only two timing precedences between the events a and d, and between the events b and c. This leads to the TSD-DL expression: $(a \rightarrow d) \sim (b \rightarrow c)$. In the sequel the second interpretation is chosen, because it describes only the essential parts of the timing precedence relation.

5.2 A Schematic Interpretation Method for Separated TSDs

One way to give an arbitrary TSD a semantics is to determine for each example TSD one TSD-DL expression. Since the number of different TSDs is limited in the ISO or CCITT documents this method is applicable. However, the better way is a schematic method which translates an arbitrary TSD into a TSD-DL expression.

This method is divided into four steps:

1. *Developing the "basic timing precedence" relation:* As a starting point every timing precedence of a TSD must be considered. There are two different kinds: First the timing precedences of events, which take place at one SAP, given by the time axis. Second the timing precedences between events at different SAPs, given by the connection of two service primitives with a solid line. The timing precedences of the TSD shown in Figure 3 are: $a \rightarrow d, b \rightarrow c, a \rightarrow b, c \rightarrow d$. For Figure 1 the following timing precedences are derived: $a \rightarrow d, b \rightarrow c, a \rightarrow b$. The tilde expresses no timing precedence and is therefore of no interest. The set of timing precedences of a TSD TS is denoted as $BP(TS)$.

2. *Construction of the "basic precedence" expression:* The basic precedence expression $BPexpr(TS)$ of a TSD represents the basic timing precedence relation expressed in TSD-DL. All elements of $BP(TS)$ are combined with the TSD-DL operator $\circledast$, which describes a conjunctive composition:
$$BPexpr(TS) \in \{bp_1 \circledast bp_2 \circledast \ldots \circledast bp_n | \{bp_1, \ldots, bp_n\} = BP(TS)\}$$

3. *Construction of the "all happens" expression:* The all happens expression $ALL(TS)$ describes the fact that all actions of a TSD have to happen. Therefore all actions of a TSD TS, whose set is denoted by $ACT(TS)$, are connected with the TSD-DL operator $\sim$:
$$ALL(TS) \in \{act_1 \sim act_2 \sim \ldots \sim act_n | \{act_1, \ldots, act_n\} = ACT(TS)\}$$

4. *Construction of the TSD-DL expression:* The TSD-DL expression, which describes the scenaric interpretation of a single TSD is the conjunctive composition of the basic precedence expression with the all happens expression:
$$(ALL(TS)) \circledast (BPexpr(TS))$$

The presented method only works with events which can be uniquely distinguished by name. The problem is a consequence of the used $\circledast$ operator which can be avoided by transformation rules. If one event is used repeatedly in one TSD, the events have to be tagged for the construction of a TSD-DL expression. These TSD-DL expressions can be simplified to expressions without the operator $\circledast$, and then the tagging can be omitted (for more details see [Fac95]).

All TSD-DL expressions derived from single TSDs of a service are alternatively combined: $(\mathrm{TSD}_1 \mid \mathrm{TSD}_2 \mid \ldots \mid \mathrm{TSD}_n)^{\underline{\omega}}$. Here TSD_i is an arbitrary TSD-DL expression for a single TSD.

5.3 Including Source and Direction of Service Primitives

The transformation method of Section 5.2 may lead to equivalent TSD-DL expressions for different TSDs.

Figure 5: TSDs with equivalent TSD-DL expressions

Both TSDs shown in Figure 5 are transformed into the very same TSD-DL expression $a \rightarrow b$, because the event identified by b is used for two different events. The distinction between these events is given by the different sources. This problem can simply be avoided by tagging every service primitive with information about its source and direction. We use A for the left SAP and B for the right one. The first TSD of Figure 5 is transformed to the TSD-DL expression $a_A^{In} \rightarrow b_B^{Out}$; the second TSD is transformed to $a_A^{In} \rightarrow b_A^{Out}$. To simplify the expression in the sequel this additional tagging is omitted, if only one event appears once in a single TSD.

6 Some Examples of TSD-DL Expressions

In this section some examples for the semantics of TSDs and transformations of TSDs into TSD-DL expressions are given. First the semantics of single TSDs is shown, secondly examples for combined TSDs are presented.

The corresponding TSD-DL expression for the complete interpretation of the TSD shown in Figure 2 is: $(a \rightarrow b)^{\underline{\omega}}$ The semantics of this expression is:

$$\begin{aligned} P_{(a\rightarrow b)^{\underline{\omega}}}(t) = \exists ns : Trace\ Nat.\ \#ns = \infty \wedge \forall n : nat.\ & P_{(a\rightarrow b)}(filter(n, ns, t)) \\ & \vee filter(n, ns, t) = \epsilon \end{aligned}$$

Here $P_{(a\rightarrow b)}(t)$ can be simplified by elimination of the existential quantifier:

$$\begin{aligned} P_{a\rightarrow b}(t) &= \exists s_1, s_2 : Trace\ Event.\ (t = s_1 \circ s_2) \wedge (s_1 = <a>) \wedge (s_2 = <b>) \\ &= (t = <a, b>) \end{aligned}$$

This leads to:

$$\begin{aligned} P_{(a\rightarrow b)^{\underline{\omega}}}(t) = \exists ns : Trace\ Nat.\ \#ns = \infty \wedge \forall n : nat. & filter(n, ns, t) = <a, b> \\ & \vee filter(n, ns, t) = \epsilon \end{aligned}$$

To determine the accepted traces first finite traces are examined. $P_{(a\rightarrow b)^{\underline{\omega}}}(<a, b, a, b>)$ holds. This can be proven with $ns = <n_1, n_1, n_2, n_2> \circ ns'$. Analogously, $P_{(a\rightarrow b)^{\underline{\omega}}}(<a, a, b, b>)$ holds, because $ns = <n_1, n_2, n_1, n_2> \circ ns'$. That $P_{(a\rightarrow b)^{\underline{\omega}}}(<b, a>)$ does not hold can be proven with $ns = <n_1, n_2> \circ ns'$ by case distinction on $n_1 = n_2$.

Now we turn to the behavior of infinite traces. Let t be an infinite sequence of $<a,b>$, in a formal notation: $t = \mathsf{fix}\boldsymbol{\lambda} s.<a,b>\circ s$. $P_{(a\to b)^{\underline{\omega}}}(t)$ holds e. g. for $ns = (\mathsf{fix}\boldsymbol{\lambda} f.\boldsymbol{\lambda} n.<n,n>\circ f.n+1).1$ This is not surprising, because for every finite prefix t' of t $P_{(a\to b)^{\underline{\omega}}}(t')$ holds.

A slightly more sophisticated case is: $t = \mathsf{fix}\boldsymbol{\lambda} s.<a,a,b>\circ s$, which is an infinite repetition of $<a,a,b>$. Informally for every event a there exists an event b, which takes place later. This can be shown with $ns = (\mathsf{fix}\boldsymbol{\lambda} f.\boldsymbol{\lambda} n.<2*n, 2*n+1, n>\circ f.n+1).1$. From a practical point of view it is not surprising that $P_{(a\to b)^{\underline{\omega}}}(\mathsf{fix}\boldsymbol{\lambda} s.<a,a,b>\circ s)$ holds, because for every a there exists exactly one b which happens later. Therefore, the liveness part of the TSD is fulfilled.

That $P_{(a\to b)^{\underline{\omega}}}(\mathsf{fix}\boldsymbol{\lambda} s.<a,b,b>\circ s)$ does not hold can be shown with $ns = <n_1,n_2,n_3> \circ ns'$ and case distinction.

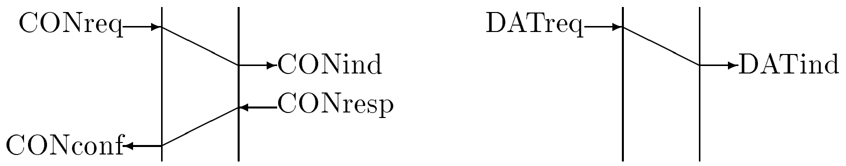

Figure 6: Example TSDs

The TSD-DL expression for the TSDs shown in Figure 6, is after some simplification:

$$(((CONreq \to CONind) \to (CONresp \to CONconf))|(DATreq \to DATind))^{\underline{\omega}}$$

Now

$$[\![(((CONreq \to CONind) \to (CONresp \to CONconf))|(DATreq \to DATind))^{\underline{\omega}}]\!]_{\mathcal{TSD}}$$

describes the set of all traces which are correct with respect to the TSDs presented in Figure 6. This example demonstrates that the presented representation can be used also to describe the TSD part of a real layer specification.

7 Extensions to TSD-DL

With TSD-DL the TSD part of a service specification can be formally described. However, some facets of a service specification are not covered by the used graphical notation for TSDs. This is a consequence of the verbal additions in the ISO or CCITT documents. In this section, some extensions of TSDs are presented. With these extensions it is possible to describe the TSD part of an OSI service formally, including some features like a formalization of connection disruption or the service primitive's parameter. We name our extension *time sequence diagram description language extended* (TSD-DLext).

7.1 Formalization of Connection Disruption

Many original service specifications using TSDs are not able to deal correctly with data-loss in the case of a connection disruption, or they handle it with an informal addition.

An example for this is the INRES service [Hog89, Hog91], which has been developed to represent a simple example service. The motivation for using the INRES service instead of a concrete ISO/OSI service is that for the INRES service there exists an informal and a formal specification. The latter is used to avoid misinterpretations. In [Hog89, Hog91] the data transfer of the INRES service is informally specified with the TSD presented in Figure 7.

Figure 7: Data transfer in the INRES service

In the TSDs presented in Figure 7 every *data request* (DATreq) leads to a *data indication* (DATind) or *disconnect indication* (DISind). For example, suppose a service user issues five DATreqs. Suppose also that only the first one is successful and leads to one DATind and then the connection is disturbed. Due to the TSDs presented in Figure 7, the service provider has then to emit four DISind. But this is in contrast to the formal specifications of the service in LOTOS, SDL and Estelle in [Hog89, Hog91], where in this case only one DISind has to be sent. Therefore, in [BHS91] an additional TSD that we show in Figure 8 is suggested.

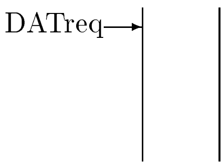

Figure 8: Extension of the data transfer in the INRES service

Using the additional TSD presented in Figure 8 it is now possible that only one DISind is sent. But then the TSDs of Figures 7 and 8 also describe an insecure data transfer too, because loss of data is possible during two successful data transfers. A first solution is to forbid this behavior in the connection part of a service specification, which describes the properties of a connection. however, this would lead e.g. in [ISO84b] to a huge number of TSDs, because in the transport layer every TSD can be interrupted. As a consequence, all possible interruptions of all TSDs have to be given in detail.

To avoid this problem a new operator for TSDs is introduced in Figure 9.

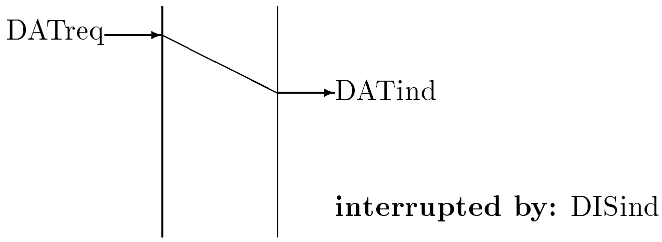

Figure 9: TSD with kill-operator

The corresponding TSD-DL expression of the TSD presented in Figure 9 is

$$(DATreq \rightarrow DATind)^{\underline{\omega}} \dagger DISind$$

with the semantics that a $DISind$ may interrupt the complete interpretation of $DATreq \rightarrow DATind$.

With the same method the verbal addition in the service definition of the transport layer in [ISO84b, CCI88e] can be totally formalized. This verbal addition describes that the transport layer service primitives T-DISCONNECTrequest or T-DISCONNECTindication may terminate any of the other sequence of service primitives.

7.2 Including Parameter of Service Primitives

Normally, service primitives are used in TSDs without parameters. However, the parameters of data transfer service primitives are important, because they are the essential part of a communication. The simple introduction of parameters leads to the TSD presented in Figure 10.

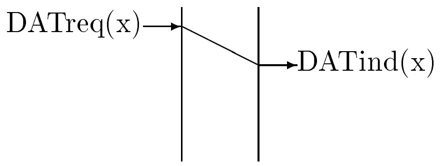

Figure 10: Data transfer with parameters

This shows a correct data transfer. It does not include the *residual failure probability* concept, in which non detectable transfer errors are possible. This concept is a part of the OSI basic reference model [ISO84a] and therefore ought to be modeled. To describe this behavior we extend Time Sequence Diagrams with *parameter restrictions*. With this concept parameters which are in a previously defined relation are describable in TSDs. For example, the TSD pictured in Figure 11 defines that $DATreq(x)$ and $DATind(y)$ belong to same TSD iff $x \approx y$ holds.

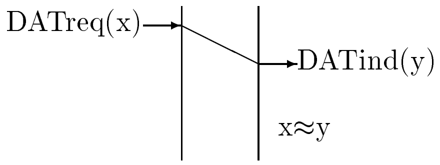

Figure 11: Data transfer with parameter restrictions

Here $\approx$ is an equivalence relation, which describes the class of non detectable errors. The TSD presented in Figure 11 can be transformed to a new TSD-DL expression

$$\textbf{Var } x, y \textbf{ With } x \approx y : DATreq(x) \rightarrow DATind(y)$$

where x and y are bound variables with the restriction that $x \approx y$ holds. Now the relation between the service primitive's parameters x and y can be expressed. The service primitives $DATreq(x)$ and $DATind(y)$ are described by the TSD of Figure 11 only if $x \approx y$ holds.

The extended TSD can also be used to describe the negotiation of quality of service parameters during the connection establishment phase. Then a partial ordering has to be used instead of the equivalence relation $\approx$.

7.3 Syntax and Semantics of TSD-DLext

Now we discuss how to syntactically and semantically model our extensions. The syntax of TSD-DLext is a simple extension of the syntax of TSD-DL with

<TSD> ::= <TSD> "†" <TSD>
<TSD> ::= **Var** <varlist> **With** <boolexpr> ":" <TSD>

where <varlist> is a list of variable identifiers separated by "," and <boolexpr> is a boolean expression.

The formal semantics of the kill operator † is:

$$\begin{aligned} P_{TS_1 \dagger TS_K}(t) = {} & P_{TS_1}(t) \\ & \vee \exists s_1, s_k, t' : Trace\ Event.\ t = s_1 \circ s_k \\ & \qquad \wedge P_{TS_K}(s_k) \\ & \qquad \wedge s_1 \neq \epsilon \\ & \qquad \wedge P_{TS_1}(s_1 \circ t') \end{aligned}$$

A correct trace for the TSD-DLext expression $TS_1 \dagger TS_K$ is a correct trace with respect to TS_1, or a trace interrupted by TS_K, where the interrupted trace is a prefix of a correct trace with respect to TS_1.

The formal semantics of a parameter restriction is defined by:

$$P_{\textbf{Var}\ x,y\ \textbf{With}\ x \approx y : TS}(t) = \exists x, y.\ x \approx y \wedge P_{TS}(t)$$

For simplicity the semantics is only for a variable list with two variables given.

8 The Formal Semantics of TSDs with Unrestricted Service User's Behavior

In the previously defined semantics no substantial distinction between input and output action is made[3]. However, if a service provider has to be specified, the behavior of the service users, who deliver the input service primitives, should not be restricted. In the previously defined semantics the occurrence of the input service primitives can be restricted by a specification as can be seen in Example 2.

Example 2: Restriction of the Service User's Behavior

In this example the restriction of the service user's behavior by the previous defined semantics is shown.

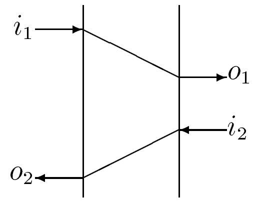

Figure 12: TSD_1

The scenaric interpretation of the TSD presented in Figure 12 is:

$$T_1 := [\![(i_1 \rightarrow o_1) \rightarrow (i_2 \rightarrow o_2)]\!]_{\mathcal{TSD}} = \{<i_1, o_1, i_2, o_2>\}$$

Let the set of all input actions be $I_1 = \{i_1, i_2, \ldots\}$, the set of all output actions be $O_1 := \{o_1, o_2, \ldots\}$. The trace $<i_2>$ is not an element of T_1. But this trace should be included, because the service provider should not be able to block a service primitive, which is received as an input. Equivalently the trace $<i_1, o_1>$ is also not an element of T_1. But it has to be, because the service provider behaved correctly and waits for the service primitive i_2, whose occurrence it can not influence. $\square$

Example 2 shows that the behavior of the service users, who form the environment of the component to be implemented, is restricted. To avoid this restriction we adopt the assumption/commitment style [AL90, AL93, SDW93, BS94] for trace sets. In [AL93] a logical presentation of the behavior of a component is given by $A \longrightarrow\!\!\triangleright\, C$. The behavior of a component is described by two parts. The assumption part A describes the necessary behavior of the environment; the commitment part C gives the rules for the component. $A \longrightarrow\!\!\triangleright\, C$ holds when either A and C hold, or when A does not hold and C holds at least until to the point in time where A does not hold.

[3]This is a consequence of the informal specification of TSDs in [ISO87a, CCI88b], where the treatment of input and output actions is not distinguished.

The previously defined $[\![TS]\!]_{\mathcal{TSD}}$ denotes the trace set in which both A and C for a given TSD TS hold. In the following we describe an extension for this trace set, where A is first violated.

There are two possibilities for inputs that violate the assumption A first:

- *false input*: After a false input a completion to a correct behavior is not possible. E.g. the trace $<i_2>$ in Example 2. Here the safety part of the assumption does not hold.
- *missing input*: These are input actions that are necessary for the completion to a correct trace. E.g. the trace $<i_1, o_1>$ in Example 2. In this case the liveness part of the assumption is violated.

The formal definition of the extensions with false input is given for a set of traces T, a set of input actions I and a set of output actions O by:

$$\begin{aligned} E_{false\_inp}(T, I, O) \quad := \{t \circ i\&x| \quad & i \in I, x \in (I \cup O)^{\omega} \\ & \wedge\ t \in PRE(T) \\ & \wedge\ t \circ <i> \notin PRE(T)\} \end{aligned}$$

PRE denotes the set of all prefixes: $PRE(T) := \{t_p|\ \exists t \in T.\ t_p \sqsubseteq t\}$. All traces with prefix $t \circ <i>$ are elements of the set of false input actions, iff t is a prefix of a correct trace of T and t appended with the input action i is not an element of the prefix set of T.

The extension regarding missing inputs is defined by:

$$\begin{aligned} E_{missing\_inp}(T, I, O) \quad := \{t| \quad & t \notin T \\ & \wedge \exists i \in I.\ t \circ <i> \in PRE(T) \\ & \wedge (\forall o \in O.\ t \circ <o> \notin PRE(T) \vee <o> \in PRE(T))\} \end{aligned}$$

A trace t is an element of the extended trace set, iff the following properties hold: First according to $t \notin T \wedge t \circ <i> \in PRE(T)$ it is guaranteed that one input action has to happen. Then it has to be ensured that every necessary and possible output action of the component has to be performed. The only output action which can occur is a *spontaneous* action, which is the first element of a TSD. Then according to the semantics of TSDs $<o> \in PRE(T)$ holds.

These definitions can be combined to a semantics for TSDs without restriction of the service user's behavior for a TSD-DL expression $Texpr$, a set of input actions I and a set of output actions O:

$$[\![Texpr]\!]^{(I,O)}_{\mathcal{TSD}} := E([\![Texpr]\!]_{\mathcal{TSD}}, I, O)$$

where

$$E(T, I, O) := T \cup E_{false\_inp}(T, I, O) \cup E_{missing\_inp}(T, I, O)$$

Some TSDs that, at a first glance seem strange, can be interpreted.

Example 3: Difficult to Implement TSD

This example demonstrates the interpretation of a TSD which is difficult to understand with respect to a later implementation.

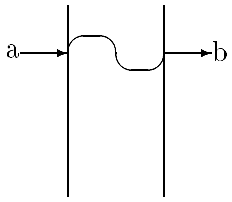

Figure 13: Example TSD

The complete interpretation of the TSD presented in Figure 13 yields the TSD-DL expression $(a \sim b)^{\underline{\omega}}$. Let $I := \{a, i\}$ and $O := \{o\}$ then: $T_1 := [\![(a \sim b)^{\underline{\omega}}]\!]_{\mathcal{TSD}} = \{t \in Trace\{a, b\} || \#a©t = \#b©t\}$

The extension regarding false inputs is defined as: $E_{false\_inp}([\![(a \sim b)^{\underline{\omega}}]\!]_{\mathcal{TSD}}, I, O) = \{t \circ i\&x | t \in PRE(T_1) \wedge x \in Trace\{a, i, b\}\}$

The extension regarding missing inputs is defined as: $E_{missing\_inp}([\![(a \sim b)^{\underline{\omega}}]\!]_{\mathcal{TSD}}, I, O) = \{t \in Trace\{a, b\} | \#a©t < \#b©t\}$

Here the interpretation of a strange behavior can be demonstrated. A correct implementation with respect to the TSD of figure 13 can provide an arbitrary number of b's. The service provider hopes that later the service user will send a fitting number of a's. If that does not happen the service user has violated the rules first. Therefore the assumption part does not hold and this trace has to be included. □

Example 3 shows an interpretation of a TSDs which is not easy to be implemented. The in this paper defined concept of missing input makes the interpretation possible. Note that the difficulties regarding the tilde symbol have as consequence that in [ISO94] it is only presented in the appendix.

9 Conclusions

In this paper a formal semantics for TSDs is developed. This gives a number of advantages. Chief among these is that the unambiguous description of a service is possible, demonstrated by our solutions of the interpretation problems of Figures 1 and 4. Because of their lack of precision, TSDs have until now been used only for commentary and overview purposes, as stated in [BHS91]. Using the semantics presented in this paper, it is possible to use TSDs as a clear and non-ambiguous description. Furthermore they can be utilized as a basis for the usage of formal methods in later development steps.

Note that our semantics is not officially justified and therefore it is only a suggestion for a formal semantics of TSDs. That a formal approach is necessary can be seen explicitly in a recent version of the TSD definition[ISO94]. In this definition the difficulties with the interpretation of $\sim$ result in putting the $\sim$ operator into the appendix of this standard. These difficulties are solved by our semantics, without restricting the service user's behavior.

In our formal semantics it is possible to state that

$$P_{(a \to b)^{\underline{\omega}}}(\mathsf{fix}\boldsymbol{\lambda} s. <a, a, b> \circ s)$$

holds. This is a major difference to most process algebraic approaches, where infinite elements can be reached only by an approximation process e.g. [BW90, Hoa85] and therefore our approach can lead to more abstract TSD descriptions.

If an executable interpretation of TSD-DL is desired, then it is possible to give a LOTOS [ISO89a] based semantics, because of the close relation of the TSD-DL operators to a process algebra. However, the introduction of the delayed choice operator [BM94] is necessary. This operator is essential for describing TSDs in a modular way. As an example let $<a, b>$ be the sequence described by one TSD and $<a, c>$ of another one. The combination of both TSDs expressed in a process algebra is:

$$(a.b) + (a.c)$$

However, then

$$(a.b) + (a.c) \neq a.(b + c)$$

holds. As a consequence the nondeterministic choice which TSD is described must be done before or at least at the time when the action a happens. If the choice has been wrong, a deadlock appears even if the correct action would follow. For example the sequence $<a, b>$ is chosen after an a occurs and then occurs the action c. To describe TSDs in a modular way the negation of the last formula should hold. To describe such a behavior in [BM94] the delayed choice operator has been introduced. Since our approach can be seen as a process algebraic one we introduced a nondeterministic operator with an equivalent behavior.

Using our approach it is possible to transform schematically one part of an arbitrary OSI service specification into a formal specification. After the formal specification of all parts of a service specification based on traces a formal development of an implementation, e.g. using stream processing functions can be achieved as in [BDD+93, DW92].

The main advantage of the schematic development of a service specification is that it is not necessary to construct specifications from scratch for every layer as in [ISO89b, ISO92]. In this paper, such a schematic development has been carried out for only one aspect concerning the TSD part. A method that covers the remaining aspects is presented in [Fac95].

The formal approach to describing TSDs shows that some special aspects of some OSI layers can not be formally expressed by TSDs. This does not lead to a problem in the ISO/CCITT documents, because they are not formal. In Section 7 some extensions of TSDs are introduced to describe these aspects. Even in a non-formal description these extensions lead to more clarity.

Acknowledgment

I thank Manfred Broy, Øystein Haugen, Max Fuchs, Konrad Slind and Ketil Stølen for careful reading preliminary versions of this paper and providing valuable feedback.

References

[AL90] Martin Abadi and Leslie Lamport. Composing specifications. Technical Report 66, Digital System Research Center, October 1990.

[AL93] Martin Abadi and Leslie Lamport. Conjoining specifications. Technical Report 118, Digital System Research Center, December 1993.

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(1):117–126, 1987.

[BDD+93] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The design of distributed systems — an introduction to FOCUS — REVISED VERSION. SFB-Bericht 342/2/92 A TUM-I9202-2, Technische Universität München, January 1993.

[BHS91] Ferenc Belina, Dieter Hogrefe, and Amardeo Sarma. *SDL with Applications from Protocol Specification.* Prentice Hall, 1991.

[BM94] J. C. M. Baeten and S. Mauw. Delayed choice: an operator for joining Message Sequence Charts. In Dieter Hogrefe and Stefan Leue, editors, *Participant's Proceedings of the Seventh International Conference on Formal Description Techniques (FORTE 94)*, pages 327–341, 1994.

[BS94] Manfred Broy and Ketil Stølen. Specification and refinement of finite dataflow networks – a relational approach. Technical Report SFB-Bericht Nr. 342/7/94 A, Technische Universität München, 1994.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[CCI88a] CCITT. X.200, reference model of Open System Interconnection for CCITT applications. Blue Book, FASCICLE VIII.4, Recommendations X.200-X.219, November 1988.

[CCI88b] CCITT. X.210, Open System Interconnection layer service definition conventions. Blue Book, FASCICLE VIII.4, Recommendations X.200-X.219, November 1988.

[CCI88c] CCITT. X.212, data link service definition of open system interconnection for CCITT applications. Blue Book, FASCICLE VIII.4, Recommendations X.200-X.219, November 1988.

[CCI88d] CCITT. X.213, network service definition of open system interconnection for CCITT applications. Blue Book, FASCICLE VIII.4, Recommendations X.200-X.219, November 1988.

[CCI88e] CCITT. X.214, transport service definition for Open System Interconnection for CCITT applications. Blue Book, FASCICLE VIII.4, Recommendations X.200-X.219, November 1988.

[DW92] C. Dendorfer and R. Weber. From service specification to protocol entity implementation – an exercise in formal protocol development. In R.J. Linn and M.Ü. Uyar, editors, *Protocol, Specification, Testing and Verification, XII*, pages 163–177. IFIP Transactions **C–8**, North-Holland, 1992.

[Fac95] Christian Facchi. *Methodik zur formalen Spezifikation des ISO/OSI Schichtenmodells.* PhD Thesis, Technische Universität München, 1995. Published in Herbert Utz Verlag Wissenschaft, München (ISBN 3–931327–94-9).

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Hog89] Dieter Hogrefe. *Estelle, LOTOS und SDL.* Springer-Verlag, 1989.

[Hog91] Dieter Hogrefe. OSI formal specification case study: the Inres protocol and service. Technical Report IAM-91-012, Universität Bern, 1991.

[ISO84a] ISO. ISO 7498: Information processing systems - open systems interconnection - basic reference model, 1984.

[ISO84b] ISO. ISO DP 8072: Information processing systems - open systems interconnection - transport service definition, 1984.

[ISO87a] ISO. Information processing systems - Open Systems Interconnection - service conventions. Technical Report ISO TR 8509, ISO, 1987.

[ISO87b] ISO. ISO 8348: Information processing systems - open systems interconnection - network service definition, 1987.

[ISO89a] ISO. LOTOS, a formal description technique based on the temporal ordering of observational behaviour. ISO International Standard 8807, February 1989.

[ISO89b] ISO/IEC. Information technology - Open System Interconection - LOTOS description of the session service. Technical Report ISO/IEC/TR 9571, International Organization for Standardization Geneva, 1989.

[ISO92] ISO/IEC. Information technology - telecommunications and information exchange between systems - formal description of ISO 8072 in LOTOS. Technical Report ISO/IEC/TR 10023, International Organization for Standardization Geneva, 1992.

[ISO94] ISO. Final DIS text of ISO/IEC 10731, information technology - Open Systems Interconnection - conventions for the definition of OSI services. Technical Report ISO/IEC JTC 1/SC 21 N 8604, ISO, 1994.

[LS87] Jacques Loeckx and Kurt Sieber. *The Foundations of Programm Verification.* John Wiley & Sons, 2nd edition edition, 1987.

[SDW93] Ketil Stølen, Frank Dederichs, and Rainer Weber. Assumption/commitment rules for networks of asynchronously communicating agents. Technical Report TUM-I9303, SFB-Bericht Nr. 342/2/93 A, Technische Universität München, 1993.

[VL86] Chris A. Vissers and Luigi Logrippo. The importance of the service concept in the design of data communication protocols. In *Protocol Specification, Testing, and Verification*, volume V, pages 3–17, 1986.

[VSvSB91] Chriss A. Vissers, Giuseppe Scollo, Marten van Sinderen, and Ed Brinksma. Specification styles in distributed systems design and verification. *Theoretical Computer Science*, 89:179–206, 1991.