Elektrotehniški vestnik 73(4): 207–214, 2006
Electrotechnical Review, Ljubljana, Slovenija

E-LOTOS-Based Compositional Service-Based Synthesis of Multi-Party Time-Sharing-Based Protocols

Monika Kapus-Kolar
Jožef Stefan Institute, Department of Communication Systems, Jamova 39, 1111 Ljubljana, Slovenia
E-mail: monika.kapus-kolar@ijs.si

Abstract. In an earlier paper, we proposed a LOTOS/T+-based method for compositional service-based construction of multi-party time-sharing-based protocols. In the present paper, we generalize the method to services with data, real-time constraints, iteration, exception handling, multi-process synchronization and process suspension/resumption, and adapt it to work with the standard specification language E-LOTOS enhanced with weak sequencing. We also report a minor error in the earlier method and propose a more flexible event-reporting scheme.

Key words: distributed service implementation, protocol synthesis, E-LOTOS

Kompozicionalno snovanje večpartnerskih protokolov z delitvijo časa na osnovi opisov pričakovanih storitev v jeziku E-LOTOS

Povzetek. V preteklosti smo predlagali metodo za kompozicionalno sintezo večpartnerskih protokolov z delitvijo časa na osnovi opisov pričakovanih storitev v jeziku LOTOS/T+. V pričujočem članku to metodo posplošimo na storitve s podatki, časovnimi omejitvami, ponavljanjem, obravnavanjem izjem, večprocesno sinhronizacijo in začasnimi prekinitvami procesov ter jo prilagodimo za delo s standardnim specifikacijskim jezikom E-LOTOS obogatenim s šibkim vrstenjem. Poročamo tudi o manjši napaki v originalni metodi in predlagamo bolj prilagodljivo shemo poročanja o dogodkih.

Ključne besede: porazdeljena implementacija storitev, sinteza protokolov, E-LOTOS

1 Introduction

For its users, a distributed server is a black box interacting with its environment through a set of service access points (SAPs). The behaviour of a server at its SAPs is the *service* it offers. The atomic instantaneous interactions constituting a service are its primitives (SPs).

In a more detailed view, each SAP belongs to a particular place and is there supported by a particular protocol entity (PE). If necessary, the PEs communicate over a medium, i.e. execute a *protocol* implementing the service. We limit our discussion to protocols operating over reliable media.

In [1], we proposed a method for compositional service-based construction of multi-party protocols. The method accepts and generates specifications written in LOTOS/T+ [2], a non-standard successor of LOTOS [3], a standard process-algebraic language for formal specification of concurrent and reactive systems. Unlike [2], another LOTOS/T+-based method, [1] does not address implementation of real-time service constraints, but can handle distributed conflicts. Unlike other protocol derivation methods based on LOTOS-like languages (see [1] for a list), the method of [1] generates protocols that resolve distributed conflicts between SPs through time sharing. If the transit delay of the underlying communication medium is short, this is often the optimal approach to conflict resolution [1].

In the present paper, we generalize the method of [1] to services with data, real-time constraints, iteration, exception handling, multi-process synchronization and process suspension/resumption, and adapt it to work with E-LOTOS [4, 5], a standard successor of LOTOS for specification of real-time systems. The only non-standard E-LOTOS feature we have not been able to avoid in the derived protocol specifications is weak sequencing [6]. We also report an error in [1] and propose a more flexible event-reporting scheme with plenty of space for protocol optimization.

The paper is organized as follows. There is no motivation section, for a thorough discussion on the applicability of time-sharing-based protocols can be found in [1]. Sect. 2 more precisely defines the adopted specification language, the server model, the concept of a well-formed service specification and the protocol derivation problem. Sects. 3 and 4, respectively, describe the syntactic and the semantic aspects of the proposed protocol

Received 23 August 2005
Accepted 10 October 2006

derivation method. Sect. 5 discusses the protocol derivation process. Sect. 6 concludes the paper by summarizing the proposed optimizations over [1] and [2].

2 Formalization of the Protocol Derivation Problem

2.1 The Basic Kinds of E-LOTOS Processes

An E-LOTOS process B executes a series of zero or more events. For each event, its relative execution time (RET) is measured relatively to the moment when it became logically enabled, while its absolute execution time (AET) is measured relatively to the start of the considered system. Let absolute time range over non-negative integers, with t denoting a time instant.

"**block**" denotes time block, and "**stop**" inaction.

"**null**" denotes immediate successful termination, i.e. a special urgent event δ. In its generalization "$P := E$", the event matches pattern P with the value of expression E (if its computation is successful), thereby updating the variables bound by the pattern. The simplest E is a constant K. Another generalization of "**null**" is "**wait**(E)", denoting a δ with RET E.

An "**i**" specifies an anonymous urgent internal process action **i** followed by a δ.

A "$GP_1@P_2[E]$" specifies an interaction of the specified process at gate G, i.e. an observable action o, followed by successful termination. Patterns P_1 and P_2, respectively, denote the data associated with the action and its RET. E is an additional constraint on the data and the RET. In E-LOTOS, an o is by definition non-urgent, i.e. always has passing of time as a legal alternative. In LOTOS/T+, an o becomes urgent as soon as it reaches its deadline. This is an important semantic difference between the two languages.

Urgent issuing of a signal X carrying data E can be specified as exception raising "**raise** $X(E)$" or, if followed by a δ, by "**signal** $X(E)$".

A "**trap exception** $X_1(IPL_1)$ **is** B_1 **endexn**... **exception** $X_n(IPL_n)$ **is** B_n **endexn exit** P **is** B_{n+1} **endexit in** B_{n+2} **endtrap**" denotes process B_{n+2} possibly followed by handling of a particular event trapped in it. Each X_i denotes a signal trapped as an exception and transferring control and data IPL_i from B_{n+2} to B_i, while "**exit** P **is** B_{n+1} **endexit**" specifies that δ in B_{n+2} transfers control and data P to B_{n+1}. A shorthand for the case where only δ is trapped and all the output data of B_{n+2} is passed to B_{n+1} is "$B_{n+2}; B_{n+1}$".

A "**loop** B **endloop**" basically denotes an infinite sequence "$B; B; \ldots$", but if an exception "**inner**" occurs in a B, this is successful termination of the loop.

A "$B_1[]B_2$" denotes a process behaving as B_1 or as B_2, where the choice is made upon the first event.

A "**choice** $P[]B$ **endch**" denotes the choice between multiple processes B, with an instance of B for every value matching the pattern P.

A "$B_1[X > B_2$" denotes process B_1 repeatedly suspended upon the start of B_2. Whenever signal X occurs in B_2, B_1 is resumed and B_2 reset to its initial state. A B of the form "$B_1[X > B_2$" contains a family of implicit trapping operators, because whenever B_2 executes its first event and becomes a B_2', B is reduced to "**trap exception** X **is** $B_1[X > B_2$ **in** B_2' **endtrap**". "$B_1[> B_2$" is a shorthand for the case with no X in B_2. Note that the $(n+1)$-th instance of B_2 is enabled when B_1 is resumed after being suspended by the n-th instance of B_2.

A "**par** $G_1\#K_1, \ldots, G_n\#K_n$ **in** $[\Gamma_1] \rightarrow B_1|| \ldots ||$ $[\Gamma_m] \rightarrow B_m$ **endpar**" denotes parallel composition of processes B_1 to B_m. Each B_i is associated with a Γ_i listing the gates on which B_i synchronizes with its peers. If the gate G on which a synchronization occurs has its synchronization degree K defined in the list "$G_1\#K_1, \ldots,$ $G_n\#K_n$", the event is a synchronization of exactly K processes B_i with G in Γ_i, otherwise it is a synchronization of all such processes. The composite process successfully terminates when all its constituents do. A shorthand for processes B_1 and B_2 synchronized on gates G_1 to G_n is "$B_1|[G_1, \ldots, G_n]|B_2$", with "$B_1|||B_2$" and "$B_1||B_2$" shorthands for the minimal and the maximal synchronization, respectively.

A "**par** P **in** $N|||B$ **endpar**" denotes independent parallel composition of multiple processes B. There is an instance of B for every value which matches the pattern P and is in the list N.

A "**rename gate** $G_1(IPL_1)$ **is** $G_1'P_1 \ldots$ **gate** $G_m(IPL_m)$ **is** $G_m'P_m$ **signal** $X_1(IPL_1')$ **is** $X_1'E_1$ ... **signal** $X_n(IPL_n')$ **is** $X_n'E_n$ **in** B **endren**" denotes process B with some of its events renamed as specified.

A "**hide** $G_1 : T_1, \ldots, G_n : T_n$ **in** B **endhide**" denotes process B with all its actions on gates G_1 to G_n of the respective types T_1 to T_n changed into **i**.

A "**var** $V_1 : T_1 := E_1, \ldots, V_n : T_n := E_n$ **in** B **endvar**" denotes process B with some variables V_i, respectively of type T_i and initialized to E_i.

A "**case** $(E_1, \ldots, E_m)$ **is** $P_1[E_1'] \rightarrow B_1| \ldots |P_n[E_n']$ $\rightarrow B_n$ **endcase**" denotes the first B_i in the list of processes B_1 to B_n for which "$(E_1, \ldots, E_m)$" matches pattern P_i and satisfies constraint E_i'. A shorthand for a series of binary decisions is "**if** E_1 **then** B_1 **elseif** E_2 **then** B_2 ... **elseif** E_n **then** B_n **else** B_{n+1} **endif**".

In the above constructs, many parts are just optional, with defaults defined in [4]. Parentheses may be used to direct parsing. We will typically refer to individual specification parts by their generic syntactic form, although in examples, we will also use shorthands or omit parts irrelevant for the discussion.

For a process B, let $\mathcal{G}(B)$ denote its visible gates. Two processes are considered equivalent if they have the same influence on every environment synchronized on their visible gates and trapping their signals and δ.

2.2 Weak Sequencing

When successful termination or an exception of a B_1 is trapped and handled by a B_2, the standard E-LOTOS semantics prescribes that B_2 starts strictly after the termination of B_1. In a real-time protocol, however, it might be crucial that a particular o in B_2 is enabled as soon as B_1 becomes able to proceed towards the particular kind of termination executing exclusively actions which o is allowed to overtake. In other words, weak sequencing might be necessary, and can indeed be introduced into E-LOTOS in a consistent and efficient way [6]. An accelerated o might resolve a choice.

Example 1 *Suppose that in "$(a[]b);c$", c is allowed to overtake a, but not b. Hence, c may occur as the first event of the process, but that resolves the choice in favour of a, for otherwise the illegal "$c;b$" would be a possible run.*

Whenever we want weak sequencing for the trappings introduced with a particular process composition operator, we will decorate the operator with a C listing the pairs (G_1, G_2) such that actions on G_2 are allowed to overtake actions on G_1, e.g. "$\mathbf{trap} \ldots \mathbf{exception}\ X(IPL)|C$ **is** B_2 $\mathbf{endexn} \ldots \mathbf{in}\ B_1\ \mathbf{endtrap}$", "$B_1;C|\ B_2$", "**loop** $C|B$ **endloop**", "$B_1[X|C>B_2$".

2.3 Server Model

We assume that a distributed server interacts with its users through a set of service gates S from a universe $\mathcal{S}$, respectively of type T_S. Every action on a service gate is considered to be an SP, even if it is dummy and thus hidden from service users. The hidden SPs are urgent, i.e. executed as soon as possible, the others are not.

Each S is located at a particular place. There are at least two places. Let p and p' denote two different places. At each place p, there is a process PE_p, the protocol entity of the place. The remaining process of the server is the communication medium.

Each PE_p has three kinds of gates: 1) It controls the local service gates. 2) For every local S and remote p', there is a transmission gate $\mathbf{s}_{T_S}^{p,p'}$, of type (T_S, T'_S), where T'_S has the same structure as T_S, except that all items in it are boolean. 3) For every S at a remote p', there is a reception gate $\mathbf{r}_{T_S}^{p}$, of type T_S.

For a local S, a PE_p can transmit a type T_S message Msg to a $PE_{p'}$ by executing an "$\mathbf{s}_{T_S}^{p,p'}(Msg, Sel)$" where every item in Msg has a corresponding selector in Sel. Only the items whose selector is true are transferred to p', together with information on T_S.

The medium delivers the message to p' after a transit delay not greater than a known $d^{p,p'}$ negligibly short from the point of the expected service users, where $(d^{p,p} = 0)$ and $(d^{p,p'} \leq d^{p,p''} + d^{p'',p'})$ for every p''. Once received by p', the message waits in a local buffer, that is formally also a part of the medium, until claimed by $PE_{p'}$ on gate $\mathbf{r}_{T_S}^{p'}$. However, Msg is not delivered to $PE_{p'}$ in its original form, as the medium replaces all its non-transferred parts with wildcards.

Actions involving the medium are hidden from service users and as such executed as soon as possible. The medium issues no signals. Messages in input buffers can be claimed in any order.

2.4 Well-Formed Service Specifications

To simplify protocol synthesis, we restrict our focus to well-formed service specifications (WFSS), i.e. to unambiguously parsable specifications in the language from Sect. 2.1 complying to the restrictions below. A WFSS is supposed to describe a non-parameterized non-blocking process Srv in which every event is a (possibly hidden) SP.

Restriction 1 *1) Every expression E in Srv must be such that its evaluation always successfully terminates. 2) Srv must be a non-parameterized "$\mathbf{rename}\ R\ \mathbf{in\ hide}\ H\ \mathbf{in}\ Srv'\ \mathbf{endhide\ endren}$" where R specifies the desired local renamings of SPs into SPs, H specifies the desired hidings of SPs, and Srv' refers exclusively to service gates. 3) No B in Srv' may be able to block without previously successfully terminating or raising an exception. 4) "$\mathbf{rename}\ldots$" is not allowed in Srv'. 5) Every "$\mathbf{loop}\ldots$" in Srv' must be such that it never successfully terminates. 6) Every "$\mathbf{case}\ldots B_1 \ldots B_n\ \mathbf{endcase}$" in Srv' must be such that it never fails to select a B_i. 7) Every event in Srv' must be a visible SP, hence*

Restriction 2 *In Srv', 1) "$\mathbf{signal}\ldots$", "$\mathbf{i}$" and "$\mathbf{hide}\ldots$" are forbidden, 2) for every δ or exception specified, there must be a trap, 3) in every "$B_1[]B_2$", the starting events of B_1 and B_2 must be SPs, 4) in every "$\mathbf{choice}\ P[]B_1\ \mathbf{endch}$", the starting events of B_1 must be SPs, 5) in every "$B_1[X > B_2$", the events of B_1 and the starting events of B_2 must be SPs, and 6) in every "$\mathbf{par}\ldots B_i \ldots$", every event of B_i must be an SP or a δ, where 3) to 6) preclude implicit $\mathbf{i}$ [4].*

A process is aware of time if upon every action, it is aware of its AET. In E-LOTOS, timing constraints can directly refer only to RETs, therefore we need

Restriction 3 *Srv' must be a "$Cl||Mn$" where 1) every T_S denotes a record with the first field of type* time*, 2) in Mn, there is no "@P" or "$\mathbf{wait}\ldots$", and 3) Cl is just a constraint [7] securing that whenever Mn executes an SP, its first data item is its AET.*

Example 2 *Suppose that $\mathcal{G}(Mn)$ is $\{a,b\}$ where T_a is a "$(\mathrm{time}, \mathrm{bool})$" and T_b is a "$(\mathrm{time}, \mathrm{nat})$". Cl can be*
"$\mathbf{var}\ \mathrm{oldt} : \mathrm{time} := 0, \mathrm{ret} : \mathrm{time}, \mathrm{newt} : \mathrm{time}\ \mathbf{in}$
$\mathbf{loop}\ (a(?\mathrm{newt}, \mathbf{any} : \mathrm{bool})@?\mathrm{ret}[\mathrm{newt} = (\mathrm{oldt} + \mathrm{ret})]\ []$
$b(?\mathrm{newt}, \mathbf{any} : \mathrm{nat})@?\mathrm{ret}[\mathrm{newt} = (\mathrm{oldt} + \mathrm{ret})]);$
$?\mathrm{oldt} := \mathrm{newt}\ \mathbf{endloop\ endvar}$".
Instead of an "$a!\mathrm{false}; b!1@!3$" in Mn, one would write "$a(?x; !\mathrm{false}); b(!(x + 3), !1)$", thereby achieving that in the presence of Cl, b is executed 3 time units after a.

Restriction 4 *Let in Mn every 1) "$\mathbf{trap}\ldots$" be a "$\mathbf{trap} \ldots \mathbf{endexn\ in}\ldots$", though "$B_1; B_2$" is also allowed, 2) "$\mathbf{exception}\ X(IPL)$" be an "$\mathbf{exception}\ X(?V_1 : T_1, \ldots, ?V_n : T_n)$", and 3) "$SP[E]$" belong to a "$\mathbf{var}\ V_1 : \mathrm{time}$*

in $SP[E]$ **endvar**" *and be an* "$S(?V_1, ?V_2, \ldots, ?V_n)[(V_1 = V_2) \wedge E']$" *of type* T_S *with* V_2 *of type* time *and* E' *not referring to* V_1.

2.5 Protocol Derivation Problem

We are looking for a mapping $\mathbf{M}_p$ which would take a WFSS and generate such PE_p that with the resulting protocol $\mathbf{M}(Srv)$, the server would be equivalent to Srv. We want that $\mathbf{M}_p$ maps a specification by mapping its parts, while noting that for an efficient protocol, the mapping has to be context-dependent.

3 Syntactic Aspects of Protocol Derivation

For a place, it might be convenient to pretend that reports on SPs belonging to different gates of Mn are exchanged through different gates. Internally, such a p would use gates $\mathbf{s}_S^{p'}$ and $\mathbf{r}_S$ instead of $\mathbf{s}_{T_S}^{p,p'}$ and $\mathbf{r}_{T_S}^{p}$, respectively. Let $\mathcal{R}$ denote the universe of gates $\mathbf{r}_S$.

Hiding and renaming of SPs without changing their location are a local matter, hence

Mapping 1 *For every p,* $\mathbf{M}_p(\textbf{rename } R \textbf{ in hide } H \textbf{ in } Srv' \textbf{ endhide endren})$ *is* "$\textbf{rename } R \cup R_p \textbf{ in hide } H \textbf{ in } \mathbf{M}_p(Srv') \textbf{ endhide endren}$", *where* R_p *renames gates* $\mathbf{s}_S^{p'}$ *and* $\mathbf{r}_S$ *into* $\mathbf{s}_{T_S}^{p,p'}$ *and* $\mathbf{r}_{T_S}^{p}$, *respectively.*

Assuming that time progresses at all places with the same speed, time awareness is a local matter, hence

Mapping 2 *For every p,* $\mathbf{M}_p(Srv')$ *is* "$Cl_p|[\mathcal{G}(\mathbf{M}_p(Mn)) \cap (\mathcal{S} \cup \mathcal{R})]|\mathbf{M}_p(Mn)$", *where* Cl_p *is just a constraint securing that whenever* $\mathbf{M}_p(Mn)$ *executes an SP or a reception, its first data item is its AET.*

A place might refer to service variables, not necessarily initializing their local copies precisely, hence

Mapping 3 *For every p, if a B in* Mn *is a* "$\textbf{var } V_1 : T_1 := E_1, \ldots, V_n : T_n := E_n \textbf{ in } B_1 \textbf{ endvar}$", $\mathbf{M}_p(B)$ *is* "$\textbf{var } V_1 : T_1 := E_{1,p}(B), \ldots, V_n : T_n := E_{n,p}(B) \textbf{ in } \mathbf{M}_p(B_1) \textbf{ endvar}$".

Rule 1 *If for a B in* Mn, *an* $\mathbf{M}_p(B)$ *refers to a non-local variable, it must be a variable visible and not local to B in* Mn.

Rule 2 *For every p and B in* Mn, *every E in* $\mathbf{M}_p(B)$ *must be unable to fail to successfully produce a result of the type implied by the surrounding context.*

Mapping 4 *For every p, if a B in* Mn *is a* "(B_1)", $\mathbf{M}_p(B)$ *is* "$(\mathbf{M}_p(B_1))$".

Mapping 5 *For every p, if a B in* Mn *is a* "**stop**" *or a* "**null**", $\mathbf{M}_p(B)$ *is B.*

A local counterpart of an update of service variables might not have to be precise, hence

Mapping 6 *For every p, if a B in* Mn *is a* "$P := E$", $\mathbf{M}_p(B)$ *is* "$P := E_p(B)$".

A local counterpart of an exception might not have to carry precisely the specified data, hence

Mapping 7 *For every p, if a B in* Mn *is a* "$\textbf{raise } X(E)$", $\mathbf{M}_p(B)$ *is* "$\textbf{raise } X(E_p(B))$", *where* $E_p(B)$ *is of the same type as E.*

The only protocol messages will be reports on individual SPs. For a report, it might be acceptable that it does not contain precise information on the SP or that the recipient does not make a precise assumption on the message contents and its arrival time. For an SP, it might be acceptable that a local counterpart not participating in its execution does not receive a message on it, but rather makes an assumption on whether or not the SP has been or will be executed and what its data and AET could be. Hence

Mapping 8 *If a B in* Mn *is an* "$S(?V_1, \ldots, ?V_n)[E]$" *with S at a p,* $\mathbf{M}_p(B)$ *is* "$(B; (|||_{\forall p'} \textbf{if } E_{1,p'}(B) \textbf{ then var } V_2' : \text{bool}, \ldots, V_n' : \text{bool} \textbf{ in } \mathbf{s}_S^{p'}((!V_1, \ldots, !V_n), (!\text{false}, ?V_2', \ldots, ?V_n'))[E_{2,p'}(B)] \textbf{ endvar else null endif}))$" *and for every* p', $\mathbf{M}_{p'}(B)$ *is* "$\textbf{if } E_{3,p'}(B) \textbf{ then } \mathbf{r}_S(?V_1, \ldots, ?V_n)[E_{4,p'}(B)] \textbf{ elseif } E_{5,p'}(B) \textbf{ then } (?V_2, \ldots, ?V_n) := E_{6,p'}(B) \textbf{ else stop endif}$", *where the identifiers* V_2' *to* V_n' *differ from those of the variables visible to B in* Mn.

Example 3 *In the following services, a, b and c belong to three different places p,* p' *and* p'', *respectively.* "$a?x[x = 1]; c; b!x; \textbf{stop}$" *does not require that a is reported to* p', *but* $\mathbf{M}_{p'}(a?x[x = 1])$ *must set the value of x at* p', *by an* "$?x := 1$". "$a?x; c; b!x; \textbf{stop}$" *requires that* p' *receives x in a report on a, because its value is not predetermined.* "$(a?x[x = 1]; B_1)[](a?x[x = 2]; B_2)$" *with both a reported to* p' *requires that x is included in the reports and that* p' *checks the received value, so that it can choose the same alternative as p. For* "$(a?x[]c?x[x = \text{false}]); \textbf{if } x \textbf{ then } b; \textbf{stop else stop endif}$", $\mathbf{M}_{p'}(c?x[x = \text{false}])$ *should be equivalent to* "**stop**".

If two cases are equivalent for a place, it need not distinguish between them, hence

Mapping 9 *For every p, if a B in* Mn *is a* "$\textbf{case } (E_1, \ldots, E_m) \textbf{ is } P_1[E_1'] \rightarrow B_1| \ldots |P_n[E_n'] \rightarrow B_n \textbf{ endcase}$", $\mathbf{M}_p(B)$ *is* "$\textbf{case } (E_{1,p}(B), \ldots, E_{m,p}(B)) \textbf{ is } P_1[E_{1,p}'(B)] \rightarrow \mathbf{M}_p(B_1)| \ldots |P_n[E_{n,p}'(B)] \rightarrow \mathbf{M}_p(B_n) \textbf{ endcase}$".

Example 4 *For* "$a?x : \text{bool}; b; \textbf{if } x \textbf{ then } (c; \textbf{stop}) \textbf{ else } (d; \textbf{stop}) \textbf{ endif}$" *with a, c and d at a p and b at a* p', $\mathbf{M}_{p'}(c; \textbf{stop})$ *and* $\mathbf{M}_{p'}(d; \textbf{stop})$ *may both be equivalent to inaction. Hence,* $\mathbf{M}_{p'}(\textbf{if} \ldots \textbf{endif})$ *could be* "$\textbf{if } \text{true} \textbf{ then stop else stop endif}$", *with x not needed at* p'.

If two subprotocols synchronize on an SP, it might be desirable that they co-operate on its reporting, hence

Mapping 10 *For every p, if a B in* Mn *is a* "$\textbf{par } D \textbf{ in } [\Gamma_1] \rightarrow B_1|| \ldots ||[\Gamma_m] \rightarrow B_m \textbf{ endpar}$", $\mathbf{M}_p(B)$ *is* "$\textbf{par } D \textbf{ in } [\Gamma_1 \cup \Gamma_{1,p}(B)] \rightarrow \mathbf{M}_p(B_1)|| \ldots ||[\Gamma_m \cup \Gamma_{m,p}(B)] \rightarrow \mathbf{M}_p(B_m) \textbf{ endpar}$", *where for every i in* $\{1, \ldots, m\}$, *there is such a set* $U_i(B)$ *of pairs* (S, p) *with S in* Γ_i, *not at p and not mentioned in D, that for every p,* $\Gamma_{i,p}(B)$ *consists of gates* $\mathbf{s}_S^{p'}$ *with* (S, p') *in* $U_i(B)$ *and of gates* $\mathbf{r}_S$ *with* (S, p) *in* $U_i(B)$.

Mapping 11 *For every p, if a B in* Mn *is a* "$\textbf{par } P \textbf{ in } N|||B_1 \textbf{ endpar}$", $\mathbf{M}_p(B)$ *is* "$\textbf{par } P \textbf{ in } N|||\mathbf{M}_p(B_1) \textbf{ endpar}$".

Example 5 *For* "$(a; b)|[a]|(a; c)$" *with a at a p and b and c at a* p', *a must be reported to* p' *both in subprotocol* $\mathbf{M}(a; b)$ *and in subprotocol* $\mathbf{M}(a; c)$, *but if* $\mathbf{M}_p(a; b)$ *and* $\mathbf{M}_p(a; c)$ *are synchronized on* $\mathbf{s}_a^{p'}$, *and* $\mathbf{M}_{p'}(a; b)$ *and* $\mathbf{M}_{p'}(a; c)$ *on* $\mathbf{r}_a$, *there is only one protocol message.*

At every place, receptions must be allowed to overtake any non-SP action, for otherwise a message might be received with a non-zero local delay. In [1], we were not aware of that and so the method, unless corrected as in [8], generates incorrect protocols. Hence

Mapping 12 *For every p, if a B in* Mn *is a "*$B_1; B_2$*",* $\mathbf{M}_p(B)$ *is "*$(\mathbf{M}_p(B_1); C_p(B)|\mathbf{M}_p(B_2))$*", where* $C_p(B)$ *lists all the pairs* (G_1, G_2) *of* G_1 *in* $(\mathcal{G}(\mathbf{M}_p(B_1)) \setminus \mathcal{S})$ *and* G_2 *in* $(\mathcal{G}(\mathbf{M}_p(B_2)) \cap \mathcal{R})$*.*

Mapping 13 *For every p, if a B in* Mn *is a "***loop** B_1 **endloop***",* $\mathbf{M}_p(B)$ *is "***loop** $C_p(B)|\mathbf{M}_p(B_1)$ **endloop***", where* $C_p(B)$ *lists all the pairs* (G_1, G_2) *of* G_1 *in* $(\mathcal{G}(\mathbf{M}_p(B_1))\setminus\mathcal{S})$ *and* G_2 *in* $(\mathcal{G}(\mathbf{M}_p(B_1)) \cap \mathcal{R})$*.*

Mapping 14 *For every p, if a B in* Mn *is a "***trap exception** $X_1(IPL_1)$ **is** B_1 **endexn**...**exception** $X_n(IPL_n)$ **is** B_n **endexn in** B_{n+1} **endtrap***",* $\mathbf{M}_p(B)$ *is "***trap exception** $X_1(IPL_1)|C_{1,p}(B)$ **is** $\mathbf{M}_p(B_1)$ **endexn**...**exception** $X_n(IPL_n)|C_{n,p}(B)$ **is** $\mathbf{M}_p(B_n)$ **endexn in** $\mathbf{M}_p(B_{n+1})$ **endtrap***", where a* $C_{i,p}(B)$ *lists all the pairs* (G_1, G_2) *of* G_1 *in* $(\mathcal{G}(\mathbf{M}_p(B_{n+1})) \setminus \mathcal{S})$ *and* G_2 *in* $(\mathcal{G}(\mathbf{M}_p(B_i)) \cap \mathcal{R})$*.*

Example 6 *For "*$a@!0; (b@!2[]c@?x[x \geq 4])$*" with a, b and c at three different places p, p′ and p″, respectively, suppose that a report on a arrives to p′ and p″ at times* 1 *and* 5*, respectively, and a report on b arrives to p″ at* 3*. If p″ implements ";" as strong sequencing, it enables* $\mathbf{M}_{p''}(b \ldots [] c \ldots)$ *at* 5*, when it suddenly becomes ready not only for reception of the report on b, but also for an illegal c. If the sequencing at p″ is adequately weakened, p″ receives the report on b already at* 3*, in time to prevent c.*

In the distributed implementation of choice, there are cases where it is necessary that a place a priori abandons all but one of the alternatives, hence

Mapping 15 *For every p, if a B in* Mn *is a "*$B_1[]B_2$*",* $\mathbf{M}_p(B)$ *is "***if** $E_{1,p}(B)$ **then** $\mathbf{M}_p(B_1)$ **elseif** $E_{2,p}(B)$ **then** $\mathbf{M}_p(B_2)$ **else** $\mathbf{M}_p(B_1)[]\mathbf{M}_p(B_2)$ **endif***".*

Mapping 16 *For every p, if a B in* Mn *is a "***choice** $P[]B_1$ **endch***", with P of a type T,* $\mathbf{M}_p(B)$ *is "***case** $E_{1,p}(B)$ **is** $P[E_{2,p}(B)] \rightarrow \mathbf{M}_p(B_1)|$**any** $: T \rightarrow$ **choice** $P[]\mathbf{M}_p(B_1)$ **endch endcase***", with* $E_{1,p}(B)$ *of type T.*

Example 7 *For "*$(a[]b); c; d$*" with a, b and c at a p and d at a p′, a and b need not be reported to p′, hence* $\mathbf{M}_{p'}(a)$ *and* $\mathbf{M}_{p'}(b)$ *can both be a "***null***", but then* $(\mathbf{M}_{p'}(a)[]\mathbf{M}_{p'}(b))$ *is "***null**[]**null***", i.e. a process unable to execute the required δ [4]. Keeping only one of the equivalent alternatives yields a correct* $\mathbf{M}_{p'}(a[]b)$*.*

Rule 3 *For every p and B of the form "*$B_1[]B_2$*", "*$B_2[]B_1$*", "***choice** $P[]B_1$ **endch***" or "*$B_2[X|C > B_1$*" in* $\mathbf{M}_p(Mn)$*, every starting event of* B_1 *must be an o, except in the last case where it may also be an X.*

No place may ever suspend transmission or reception of an SP report, for the recipient of the report might otherwise fail to detect the SP in time. Hence transmissions and receptions in an interrupted process must be allowed to overtake actions in the interrupting process. In the case of multiple consecutive instances of an interrupting process, receptions in each of them must be allowed to overtake non-SP actions in the preceding ones. In the distributed implementation of a "$B_1[> B_2$", there are cases where it is necessary that a place a priori abandons execution of B_1. Hence

Mapping 17 *For every p, if a B in* Mn *is a "*$B_1[X > B_2$*",* $\mathbf{M}_p(B)$ *is "***if** $E_p(B)$ **then** $\mathbf{M}_p(B_2)$ **else rename** $R_{1,p}(B)$ **in rename** $R_{2,p}(B)$ **in** $\mathbf{M}_p(B_1)$ **endren** $[X|(C_{1,p}(B) \cup C_{2,p}(B)) > \mathbf{M}_p(B_2)$ **endren endif***", where* $R_{2,p}$ *renames every G in* $\mathcal{G}(\mathbf{M}_p(B_1))$ *into a different G′ not in* $\mathcal{G}(\mathbf{M}_p(B_2))$*,* $R_{1,p}(B)$ *restores the original names of the gates,* $C_{1,p}(B)$ *lists all the pairs* (G_1, G_2) *with* G_1 *in* $(\mathcal{G}(\mathbf{M}_p(B_2))\setminus\mathcal{S})$ *and* G_2 *in* $(\mathcal{G}(\mathbf{M}_p(B_2)) \cap \mathcal{R})$*, and* $C_{2,p}(B)$ *lists all the pairs* (G_1, G_2') *with* G_1 *in* $\mathcal{G}(\mathbf{M}_p(B_2))$ *and* G_2 *in* $(\mathcal{G}(\mathbf{M}_p(B_1))\setminus\mathcal{S})$*.*

Example 8 *For "*$((a; \mathbf{stop})[> b); c; d$*" with a, b and c at a p and d at a p′, it is appropriate that* $\mathbf{M}_{p'}(b)$ *is equivalent to "***null***". Within an "*$\mathbf{M}_{p'}(a; \mathbf{stop})[> \mathbf{M}_{p'}(b)$*", such an* $\mathbf{M}_{p'}(b)$ *would be unable to execute the required δ [4]. Keeping just* $\mathbf{M}_{p'}(b)$ *yields a correct* $\mathbf{M}_{p'}((a; \mathbf{stop})[> b)$*.*

4 Semantic Aspects of Protocol Derivation

Let I denote an instance of a B in Mn. An I of a B of the form "$SP[E]$" is an A. For an I, let $\mathbf{M}(I)$ denote the corresponding subprotocol of $\mathbf{M}(Srv)$, i.e. the corresponding instance of $\mathbf{M}(B)$, with each $\mathbf{M}_p(I)$ denoting the corresponding instance of $\mathbf{M}_p(B)$. In the particular case of an I of the form "**loop** B **endloop**" or "$I_1[X > B$", B has an infinite series of instances, where in a corresponding "**loop** $\mathbf{M}_p(B)$ **endloop**" or "$\mathbf{M}_p(I_1)[X > \mathbf{M}_p(B)$", the instance of $\mathbf{M}_p(B)$ corresponding to the n-th instance of B is the n-th one, while in the case of an $\mathbf{M}_p(I_1[X > B)$ reduced to $\mathbf{M}_p(B)$, the process corresponds to the first instance of B.

Let s and s' denote two different SPs in Mn. For an s, let $Prt(s)$ list the participating A. An s is uniquely determined by $Prt(s)$ and by the data it carries (including, thanks to Cl, its AET). For an I, let $Prt(s, I)$ list the A in $Prt(s)$ which are subprocesses of I.

With such detailed characterization of SPs, Srv is completely deterministic, so that the problem of its distributed implementation reduces to proper implementation of its individual runs ρ, prevention of non-determinism in individual $\mathbf{M}_p(Srv)$, and securing that time constraints of individual $\mathbf{M}_p(Srv)$ and the medium suffice for proper resolution of global conflicts. The more urgent the SPs of Srv' are, the smaller is the number of the possible runs, i.e. the easier it is to satisfy the rules below, i.e. the lesser is the need for inter-place communication.

Example 9 *Take "*$a@!0; b@!1$*" with a at a p and b at a p′. If a at time* 0 *is not urgent, its invocation is not mandatory, so that, to satisfy the empty ρ, p′ must refrain from executing b at time* 1 *before receiving a report on a. If a is urgent, the empty ρ is impossible, and hence the report not necessary.*

To prevent local non-determinism, we must prevent ambiguous transitions, hence

Rule 4 *For every p,* $\mathbf{M}_p(Srv)$ *is forbidden to have a state in which two or more outgoing transitions would represent receptions with identical gate and data.*

For a ρ, let Exc^ρ list, in the order of occurrence, the executed SPs. For an I, let $Exc^\rho(I)$ be the projection of Exc^ρ onto the SPs s with a non-empty $Prt(s, I)$.

For a ρ, let Enb^ρ list the I enabled during the run. For an I in an Enb^ρ, let $At^\rho(I)$ denote the time when it gets enabled in ρ.

For an I, let $Var(I)$ list the variables visible and not local to the process. For an I in an Enb^ρ, let $In^\rho(I)$ for every V in $Var(I)$ provide its value upon enabling of I in ρ, if not undefined. For a p, let $\mathbf{M}_p^\rho(I)$ denote $\mathbf{M}_p(I)$ started with the data in $In^\rho(I)$.

For each particular ρ, the expected actions of $\mathbf{M}(Mn)$ are the following: For every s at a p at a t in Exc^ρ, it is expected that $\mathbf{M}_p^\rho(A)$ with A in $Prt(s)$ execute s and promptly report it as specified, while for every p', it is expected that $\mathbf{M}_{p'}^\rho(A)$ with A in $Prt(s)$ receive the incoming reports on s at t' with $(t \leq t' \leq t + d^{p,p'})$, i.e. immediately upon arrival, in a manner maintaining communication closedness of individual $\mathbf{M}(A)$.

For every ρ and p, we expect and will secure that $\mathbf{M}_p(Srv)$ proceeds as follows: 1) Whenever it enables an $\mathbf{M}_p(I)$ with I in Enb^ρ of the form "$\mathbf{case}\ldots\mathbf{endcase}$", "$I_1[]I_2$" or "$\mathbf{choice}\ P[]B\ \mathbf{endch}$" with an I' the alternative selected in I in ρ, $\mathbf{M}_p(I)$ is, at least virtually, executed as $\mathbf{M}_p^\rho(I')$. 2) $\mathbf{M}_p(Mn)$ executes exclusively the actions it is supposed to execute for ρ, taking care that its SPs are executed in the order specified in Exc^ρ.

To be able to ignore the fact that for a ρ, an $\mathbf{M}_p(Srv)$ proceeding as expected might enable an $\mathbf{M}_p(I)$ with I not in Enb^ρ, we introduce

Rule 5 *For every ρ and p, it must be impossible that $\mathbf{M}_p(Srv)$ proceeding as expected for ρ reaches a state in which it could allow $\mathbf{M}_p(Mn)$ to execute an s with an A in $Prt(s)$ not in Enb^ρ.*

Rule 6 *For every ρ, p, I of the form "$I_1[X > B$" and the second or a later instance I_2 of B in I, if I_2 is not in Enb^ρ, it must be impossible that $\mathbf{M}_p(Srv)$ proceeding as expected for ρ enables $\mathbf{M}_p(I_2)$ and subsequently reaches a state in which it could allow $\mathbf{M}_p(Mn)$ to execute an s with a non-empty $Prt(s, I_1)$.*

Example 10 *Take "$a; b; c$" with a and b at a p and c at a p'. It is appropriate that p' receives from p a report on b, while reporting of a is not necessary. As a is non-urgent, service users might decide not to invoke it. Executing the empty ρ, $\mathbf{M}_{p'}(a; b; c)$ operating as described enables $\mathbf{M}_{p'}(b)$ in spite of "b" not in Enb^ρ, but correctly refrains from enabling $\mathbf{M}_{p'}(c)$.*

For a ρ and a p, let Enb_p^ρ list the I in Enb^ρ with $\mathbf{M}_p(I)$ enabled while $\mathbf{M}_p(Srv)$ proceeds as expected for ρ.

Every expectedly enabled $\mathbf{M}_p(I)$ must be supplied with the expected input data, hence

Rule 7 *For every ρ, p, I in Enb_p^ρ and V in $Var(I)$, if V is an input variable of $\mathbf{M}_p(I)$, it must have a value K defined in $In^\rho(I)$ and when $\mathbf{M}(Srv)$ proceeding as expected for ρ enables $\mathbf{M}_p(I)$, the value of its input V must be K.*

For a p, let Act_p^ρ list the A which are in $Prt(s)$ of an s in Exc^ρ such that s is at p or that it is at a p' with $E_{1,p}(A)$ true in $\mathbf{M}_{p'}^\rho(A)$ after s. For an I, let $Act_p^\rho(I)$ list the A in Act_p^ρ which are subprocesses of I.

To be able to pretend that in every local decision, the place selects exactly the alternative intended for the particular ρ, we introduce

Rule 8 *For every ρ, p and I in Enb_p^ρ of the form "$\mathbf{case}\ldots\mathbf{endcase}$", $\mathbf{M}_p^\rho(I)$ must be equivalent to $\mathbf{M}_p^\rho(I')$ with I' the alternative in I selected in ρ.*

Rule 9 *For every ρ, p, I in Enb_p^ρ of the form "$I_1[]I_2$" or "$\mathbf{choice}\ P[]B\ \mathbf{endch}$", and two alternatives I' and I'' of I, if $\mathbf{M}_p^\rho(I)$ allows only $\mathbf{M}_p(I')$, I' must be the alternative of I selected in ρ, or $Act_p^\rho(I'')$ must be empty and $\mathbf{M}_p^\rho(I')$ equivalent to $\mathbf{M}_p^\rho(I'')$.*

Rule 10 *For every ρ, p and I in Enb_p^ρ of the form "$I_1[X > B$", if $\mathbf{M}_p^\rho(I)$ omits $\mathbf{M}_p(I_1)$, $Act_p^\rho(I_1)$ must be empty and $\mathbf{M}_p^\rho(B)$ unable to raise X.*

If an $\mathbf{M}_p(A)$ is supposed to perform an action, it must be timely enabled and while the action is still pending, not suspended permanently or too long, hence

Rule 11 *For every ρ, p, s at a t in Exc^ρ and A in $Prt(s)$, if A is in Act_p^ρ, it must be in Enb_p^ρ and if s is at p, it must be impossible that $\mathbf{M}_p(Srv)$ proceeding as expected for ρ fails to enable $\mathbf{M}_p(A)$ at t or earlier.*

Rule 12 *For every ρ, p, I of the form "$I_1[X > B$", consecutive instances I_2 and I_3 of B in I, s at p at a t in $Exc^\rho(I_1)$ and s' in $Exc^\rho(I_2)$ executed before s, if $Act_p^\rho(I_2)$ is non-empty, it must be impossible that $\mathbf{M}_p(Srv)$ proceeding as expected for ρ fails to enable $\mathbf{M}_p(I_3)$ at t or earlier.*

Rule 13 *For every ρ, p, I of the form "$I_1[X > B$", consecutive instances I_2 and I_3 of B in I, s in $Exc^\rho(I_1)$, s' at a t' in $Exc^\rho(I_2)$ with a non-empty $(Prt(s') \cap Act_p^\rho(I_2))$, and A in $Prt(s, I_1)$, if s is at p at t' with an $E_{1,p'}(A)$ true in $\mathbf{M}_p^\rho(A)$ after s, or if it is at a p' at a t with $(t + d^{p',p} \geq t')$ and $E_{1,p}(A)$ true in $\mathbf{M}_{p'}^\rho(A)$ after s, I_3 must be in Enb_p^ρ.*

Example 11 *Take "$B_1[X > B_2$" with B_1 an "$(a; \mathbf{stop})$", B_2 a "$(b; ((c; \mathbf{raise}\ X)[](d; e; \mathbf{stop})))$" and a to d at a p and reported to a p', the executor of e. If first p executes a, b and d and reports them to p', the next event might be reception of the report on b. Upon the event, p' suspends $\mathbf{M}_{p'}(B_1)$, but remains ready to receive the report on a. If its reception is the next event, it erroneously resolves the choice in $\mathbf{M}_{p'}(B_2)$ in favour of $\mathbf{M}_{p'}(c; \mathbf{raise}\ X)$, because the reception is legal only if $\mathbf{M}_{p'}(B_1)$ is actually resumed later on. Consequently, p' fails to receive the report on d and execute e.*

Subprotocols synchronized on a transmission gate must properly co-operate on it, hence

Rule 14 *For every ρ, s on a gate S at a p in Exc^ρ, non-empty subset α of $Prt(s)$ and destination p', if $\mathbf{M}_p^\rho(A)$ with A in α are synchronized on gate $\mathbf{s}_S^{p'}$, the value of $E_{1,p'}(A)$ after s in $\mathbf{M}_p^\rho(A)$ must be the same for every A in α and if it is true, there must be a "$V_2', \ldots, V_n'$" simultaneously satisfying $E_{2,p'}(A)$ after s in all $\mathbf{M}_p^\rho(A)$ with A in α.*

Places must not be too selective about the messages they want to receive, hence

Rule 15 *For every ρ, s at a p at a t in Exc^ρ and A in $Prt(s)$, if an $E_{1,p'}(A)$ is true in $\mathbf{M}_p^\rho(A)$ after s, $\mathbf{M}_{p'}^\rho(A)$ must be ready to receive the report on s sent by $\mathbf{M}_p^\rho(A)$ at any t' with $(t \leq t' \leq t + d^{p,p'})$, setting no restrictions on those among the fields V_2 to V_n which the medium might pass as a wildcard.*

The above rules secure that for every ρ, each $\mathbf{M}_p(Srv)$ can proceed as expected. It remains to secure that no unexpected SP occurs and that the expected SPs are executed in the expected global order.

The first measure against unexpected SPs are Rules 5 to 7. If an $\mathbf{M}_p(A)$ is enabled or resumed unexpectedly early, that might also result in an unexpected SP, hence

Rule 16 *For every ρ and p, it must be impossible that $\mathbf{M}_p(Srv)$ proceeding as expected for ρ reaches a state in which it could allow $\mathbf{M}_p(Mn)$ to execute an s at a t less than the latest $At^\rho(A)$ of A in $Prt(s)$.*

Rule 17 *For every ρ, p, I of the form "$I_1[X > B$" and consecutive instances I_2 and I_3 of B in I, it must be impossible that $\mathbf{M}_p(Srv)$ proceeding as expected for ρ suspends $\mathbf{M}_p(I_1)$ by $\mathbf{M}_p(I_2)$ and subsequently reaches a state in which it could allow $\mathbf{M}_p(Mn)$ to execute an s with a non-empty $Prt(s, I_1)$ earlier than at $At^\rho(I_3)$.*

Example 12 *Take "$(a@!2; b)|[b]|b@!1$" with urgent a at a p and b at a p'. In its only run "a", "b" in "$a@!2; b$" is enabled at 2. If $\mathbf{M}_{p'}(a!2)$ immediately terminates, $\mathbf{M}_{p'}(b)$ is enabled already at time 0 and can, without violating Rule 5, erroneously execute b in co-operation with $\mathbf{M}_{p'}(b@!1)$.*

Example 13 *Take "$((a@!2[]b@!5); \mathbf{stop})[X > (c@!0; d@!3; \mathbf{raise}\ X)$" with a at a p, b to d at a p', and d urgent. After c, d must also be reported to p, because otherwise $\mathbf{M}^\rho_p(a@!2)$ might be, without violating Rule 6, resumed in time to execute a.*

Timely detection of remote decisions is also important for prevention of unexpected SPs, hence

Rule 18 *For every ρ, I in Enb^ρ of the form "$I_1[]I_2$" or "$\mathbf{choice}\ P[]B\ \mathbf{endch}$", and two different alternatives I' and I'' of I, if $Exc^\rho(I'')$ is non-empty, with the first SP in it at a p at a t, then for every p', it must be impossible that $\mathbf{M}_{p'}(Srv)$ proceeding as expected for ρ reaches a state in which it could allow $\mathbf{M}_{p'}(Mn)$ to execute an s with a non-empty $Prt(s, I')$ at t or later.*

Rule 19 *For every ρ, I in Enb^ρ of the form "$I_1[X > B$" and consecutive instances I_2 and I_3 of B in I, if $Exc^\rho(I_2)$ is non-empty, with the first SP in it at a p at a t, then for every p', it must be impossible that $\mathbf{M}_{p'}(Srv)$ proceeding as expected for ρ without executing an o in $\mathbf{M}_{p'}(I_2)$ reaches a state in which it could allow $\mathbf{M}_{p'}(Mn)$ to execute an s at a t' with a non-empty $Prt(s, I_1)$ and $(t' \geq t)$, unless I_3 is in Enb^ρ with $(At^\rho(I_3) < t')$.*

Example 14 *In the run "$c; a; b; e$" of "$(c@!0; d@!3; \mathbf{stop})[> (a@!1; b?x; e!x@!2)$" with a and urgent b at a p, c to e at a p', and $d^{p,p'}$ equal to 1, a must be detected by p'. However, reporting of a is not necessary, because it is sufficiently early if p' detects the disabling upon receiving a report on b, that is unavoidable because p' must receive x.*

For a ρ, s and s' in $Exec^\rho$, A in $Prt(s)$ and A' in $Prt(s')$, let $Grd^\rho(A, A')$ be true if s' is executed after s and for a superprocess I_1 of A and a superprocess I_2 of A', there is an I of the form 1) "**trap**... **is** I_2 **endexn**... **in** I_1 **endtrap**", 2) "$I_1; I_2$", 3) "**loop** B **endloop**" or "$I'[X > B$" with I_1 and I_2 two different instances of B, or 4) "$I_2[X > B$" with I_1 an instance of B. For a ρ and s and s' in Exc^ρ, let $Grd^\rho(s, s')$ be true, i.e. s guards s', if $Grd^\rho(A, A')$ is true for an A in $Prt(s)$ and an A' in $Prt(s')$, or if Exc^ρ contains an s'' with $Grd^\rho(s, s'')$ and $Grd^\rho(s'', s')$.

For a ρ, s and s' in $Exec^\rho$, A in $Prt(s)$ and A' in $Prt(s')$, let $Chn^\rho(A, A')$ be true if $Grd^\rho(A, A')$ and A is in Act^ρ_p with p the place of s'. For a ρ and an s and an s' in Exc^ρ, let $Chn^\rho(s, s')$ be true, i.e. there is a causal chain implemented from s to s', if $Chn^\rho(A, A')$ is true for an A in $Prt(s)$ and an A' in $Prt(s')$, or if Exc^ρ contains an s'' with $Chn^\rho(s, s'')$ and $Chn^\rho(s'', s')$.

Time constraints might not be sufficient for ordering a pair of SPs, hence

Rule 20 *For every ρ and s and s' in Exc^ρ with the same AET, $Grd^\rho(s, s')$ requires $Chn^\rho(s, s')$.*

Example 15 *To implement for the run "$a; b; c$" of "$(a; b)|[b]|(b; c)$" with a and c at a p and b at a p' a chain from a to c, a must be reported to p' and b must be reported to p. If a and b are urgent, the chain serves exclusively for ordering of SPs with the same AET, otherwise also for informing that the immediate guard has actually occurred.*

Example 16 *Take "$B_1|[b, e]|B_2$" with B_1 an "$((a@!0; b@!1; \mathbf{stop})[X > (c@!1; e@!0; \mathbf{raise}\ X))$", B_2 an "$(e; b; d; e)$", a to d at a p and e at a p'. In the run "$a; c; e; b; d$", proper sequencing of e and b at time 1 requires a chain. We are free to decide whether p' should report e to p in $\mathbf{M}(B_1)$, in $\mathbf{M}(B_2)$ or in both.*

5 The Protocol Derivation Process

In [9], we prove that given a WFSS and satisfying all the above rules, one obtains a correct implementation of the specified service. However, unlike most of the earlier papers on protocol derivation, we propose no mechanical procedure for translating a WFSS into a protocol, i.e. for choosing the parameters of $\mathbf{M}$.

The reason is two-fold. As first, it is often the case that for a given service, a given partition of its SPs and a given communication medium, the given constraints, i.e. the restrictions on the service structure and the rules on the protocol parameters, cannot be satisfied simultaneously, particularly if there are many distributed conflicts and strong time constraints. The necessary compromises on the service, the partition and the medium require intervention of a designer. As second, the given constraints can typically be satisfied in many different ways, with no one indisputably better than the others. Hence it is advisable to have the constraints handled by a general-purpose constraint solver able to accept additional specific optimisation criteria.

Besides in the quality of the derived protocol, one is typically interested also in the complexity of the derivation process. For that reason, we have carefully avoided constraints referring to the global state space of the distributed server, referring exclusively to properties of Srv, of individual sequences of selectively reported SPs, and of individual $\mathbf{M}_p(Srv)$ executing their local counterparts.

Another measure for decreasing the complexity would be to intentionally reason in a highly compositional way, i.e. to take care not only that $\mathbf{M}(Srv)$ correctly implements Srv, but also that individual $\mathbf{M}(I)$ in isolation properly implement I. This is the usual approach in protocol derivation, but as such reasoning is less context-conscious, it might result in a less optimal protocol or even in an erroneous conclusion that no feasible protocol exists for the given setting.

An important step towards an efficient protocol can be restructuring of Srv without changing its external behaviour, for example

- changing the degree to which an inherent parallelism is made explicit,
- changing the location of a hidden SP (e.g. to localize a causal relation or a conflict),
- enhancing an SP with hidden parameters more accurately reflecting its identity (the information might be needed in a report on the SP),
- making a process use a copy of a variable instead of its original, with the copy generated at a carefully chosen point (one can thereby control the point of sending the data to a place needing it),
- introducing a dummy SP marking completion of a group of SPs at a particular place, so that completion of the group can be reported by reporting the dummy (facilitates the protocol optimization proposed in [10]), or
- changing termination of a process into a dummy SP indicating that the process is ready for disabling, with an auxiliary process detecting the SP and performing the disabling (facilitates the protocol optimization proposed in [11], helpful particularly for processes comprising terminating and non-terminating alternatives or alternatives with different termination events).

Ideally, the constraint solver would co-operate with an expert system optimizing the structure of Srv.

6 Concluding Remarks

The proposed method generates protocols securing that any SP legal during a particular time interval is available to service users through the entire interval, while [2] and [1] strive only for occasional availability. The method accepts a much wider class of service processes. Another contribution is a flexible SP-reporting scheme with plenty of space for protocol optimization, particularly for message re-use and prevention of duplicate causal chains and unnecessary reports on urgent SPs and remote decisions.

7 References

[1] M. Kapus-Kolar, "Compositional service-based construction of multi-party time-sharing-based protocols," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E86-A, no. 9, pp. 2405–2412, 2003.

[2] A. Nakata, T. Higashino, and K. Taniguchi, "Protocol synthesis from timed and structured specifications," *Proc. ICNP'95*, pp. 74–81, IEEE Computer Society, 1995.

[3] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 25–59, 1987.

[4] ISO/IEC, *Enhancements to LOTOS (E-LOTOS)*, ISO/IEC 15437, ISO – Information Technology, 2001.

[5] A. Verdejo, *E-LOTOS: Tutorial and Semantics*, M.S. thesis, Universidad Complutense de Madrid, 1999.

[6] M. Kapus-Kolar, "Towards weak sequencing for E-LOTOS," *Computer Standards & Interfaces*, vol. 28, no. 1, pp. 59–73, 2005.

[7] C. A. Vissers, G. Scollo, M. van Sinderen, and H. Brinksma, "Specification styles in distributed systems design and verification," *Theoretical Computer Science*, vol. 89, pp. 179–206, 1991.

[8] M. Kapus-Kolar, *A Correction to Compositional Service-Based Construction of Multi-Party Time-Sharing-Based Protocols*, Jožef Stefan Institute tech. rept. #8896, 2003.

[9] M. Kapus-Kolar, *E-LOTOS-Based Compositional Service-Based Synthesis of Multi-Party Time-Sharing-Based Protocols*, Jožef Stefan Institute tech. rept. #9200, 2006.

[10] F. Khendek, G. von Bochmann, and C. Kant, "New results on deriving protocol specifications from service specifications," *Proc. SIGCOMM'89*, pp. 136–145, ACM, 1989.

[11] M. Kapus-Kolar, "More efficient functionality decomposition in LOTOS," *Informatica*, vol. 23, no. 3, pp. 259–273, 1999.

Monika Kapus-Kolar received her B.Sc. degree in electrical engineering from the University of Maribor, Slovenia, and the M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia. Since 1981 she has been with the Jožef Stefan Institute in Ljubljana. Her current research interests include formal specification techniques and methods for distributed systems development.