Elektrotehniški vestnik XX(Y): ??–??, YEAR
Electrotechnical Review, Ljubljana, Slovenija

Service-Based Synthesis of Two-Party Protocols

Monika Kapus-Kolar

Dept. of Digital Communications and Networks, Jožef Stefan Institute,
POB 3000, SI-1001 Ljubljana, Slovenia

Abstract. The paper addresses the problem of implementing a service on a distributed server consisting of two protocol entities communicating over a pair of reliable, unbounded, first-in-first-out channels. It is demonstrated that for the adopted server type, the problem of constructing a well-formed service (WFS), i.e. a service for which a protocol can be derived, reduces to the problem of non-service-based protocol synthesis, for which several methods already exist. Guidelines are given for easy and efficient tool-supported WFS synthesis. In particular, the optimistic and the pessimistic approach are confronted. Finally, a method is proposed for automatic derivation of a protocol implementing a given WFS with a minimum number of protocol messages and with exactly the desired degree of concurrency. Services and protocols are modelled by finite state machines with additional annotations. The problem of compositional service specification and implementation is also address.

Key words: service synthesis, distributed service implementation, automated protocol derivation

Snovanje dvostranskih protokolov na podlagi pričakovanih storitev

Povzetek. Članek obravnava problem izvedbe storitev na porazdeljenem strežniku, sestavljenem iz dveh protokolnih osebkov, ki komunicirata prek para zanesljivih neomejenih asinhronih kanalov. Za privzeti tip strežnika pokažemo, da se problem snovanja dobro oblikovane storitve, to je storitve, za katero lahko izpeljemo ustrezen protokol, prevede na problem snovanja protokola brez upoštevanja pričakovane storitve, za katerega že obstajajo številne metode reševanja. Podani so napotki za preprosto in učinkovito snovanje dobro oblikovanih storitev s pomočjo orodij, pri čemer primerjamo optimističen in pesimističen pristop. Predlagamo, kako naj se za izvedbo podane dobro oblikovane storitve avtomatično izpelje protokol z minimalnim številom protokolnih sporočil in z želeno stopnjo vzporednosti izvajanja. Obravnavamo tudi problem kompozicijskega snovanja storitev in njihovih izvedb.

Ključne besede: snovanje storitev, porazdeljena izvedba storitev, avtomatizirana izpeljava protokolov

1 Introduction

Information technology users increasingly depend on *distributed (virtual) servers*. In an abstract view, a distributed server is a black box interacting with its users through a set of service-access points (SAPs), in the following also called 'places'. Its behaviour at SAPs is the *service* it offers. In a more detailed abstract view, each individual SAP is supported by a protocol entity (PE). If necessary, PEs co-ordinate their work by communicating over internal server channels. The overall activity of the PEs is referred to as the *protocol* implementing the service.

Quick introduction of new services requires systematic derivation of the supporting protocols. [8] gives a very exhaustive survey of the numerous existing *protocol synthesis* methods, differing in the *server architecture* and in the *specification formalism* for which they are intended. In our paper we discuss protocol derivation for servers consisting of two PEs communicating over a pair of reliable, unbounded, first-in-first-out (FIFO) channels. The adopted formal specification technique (FDT) are *finite state machines* (FSMs), because our experience shows that people keep preferring them in spite of being constantly pressed by theoreticians to divert to more powerful FDTs.

A FSM models a process as sequentially proceeding from a state to a state, the transitions representing various actions. Unfortunately, FSMs are a very low-level model of process activities, even if the specified actions are extended with parameters. Their main deficiency in protocol synthesis is that they model concurrency of actions as their interleaving,

Received
Accepted

i.e. their execution in various orders is specified as a set of alternative paths. Hence, it might happen that two actions are specified in a sequence or as alternatives, but actually intended for concurrent execution. The main contribution of our work is an advice how to (with the help of additional information) derive protocols implementing exactly the desired level of concurrency. We provide guidelines that seem as simple and close to the human intuition as the FSM model itself. The paper is an excerpt from [4].

2 Communication Model

Let a distributed server (Fig. 1) consist of PEs PE_1 and PE_2 communicating over a pair of reliable, unbounded, initially empty FIFO channels with an unpredictable, but finite transit delay. Let $-m$ and $+m$, respectively, specify transmission and reception of a protocol message m.

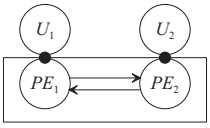

Figure 1. The distributed server architecture.

Let p and p' denote two different elements of $\{1,2\}$. A U_p denotes the service user served by PE_p. They are supposed to interact synchronously, i.e. via instantaneous common actions – service primitives.

Let variables a, b, c, i range over individual service actions and x over sequences of actions. Where necessary, a subscript p shall indicate their executor PE_p. An a_p might be a service primitive executed for U_p, or an internal action of PE_p, i.e. an action that is not directly visible to U_p, but serves some purpose, for example manipulates local data or makes a decision.

FSMs FSM_0 and FSM_p, respectively, specify the system's service and the behaviour of PE_p. They are supposed to be deterministic, i.e. no state is allowed to have two or more equally labelled outgoing transitions and only states with no outgoing transitions are legal final states. For any FSM, we are only interested in the behaviour it represents, while states are for our purposes only an auxiliary, abstract concept. Hence *fold and unfold FSMs as convenient.*

3 Service Synthesis as Protocol Synthesis

All that a PE_p does to implement a service can be described as executing local actions, reporting them to $PE_{p'}$ and receiving reports from $PE_{p'}$. *Any protocol message can be interpreted as a consequence of a local action executed by the sender.* In [9], a protocol message transmission $-a_p$ is always specified by mapping a transition a_p in FSM_0 into a sequence a_p-a_p in FSM_p. Following that approach, *introduction of a $-a_p$ into FSM_p can be induced by explicitly specifying a_p in the appropriate point of FSM_0.* A proper protocol-derivation procedure will generate the $-a_p$ only where necessary, but a designer must ensure that FSM_0 is a *well-formed service specification* (WFS), i.e. that at least some protocol exists for it.

A FSM_0 is a WFS if it can be implemented by a *best-effort protocol* (BEP). For the adopted channel type, a BEP requires that any a_p is immediately reported. Knowing that a $-a_p$ is in FSM_p always immediately preceded by a_p, the specification of the a_p can be made implicit, i.e. the a_p omitted from FSM_p. As an additional simplification, constructing a BEP, one can entirely drop the '+' and '-' signs, for we know that an a_p always represents a $-a_p$ if in FSM_p, and a $+a_p$ if in $FSM_{p'}$. Consequently, we have for any BEP $FSM_1 = FSM_2 = FSM_0$ and

Lemma 1 *[4] A FSM_0 is a WFS iff the protocol in which both PEs behave as specified by FSM_0 is syntactically correct (i.e. without non-executable actions, unspecified receptions and deadlocks).*

In other words, *WFS construction reduces to non-service-based protocol construction*, for which various methods already exist (e.g. [11, 2]).

4 Optimistic versus Pessimistic Approach to Service Synthesis

4.1 Optimistic Approach

The usual approach to WFS construction is *optimistic* (OA). One represents all the desirable sequences of service actions in a FSM_0 and hopes that this is already a WFS, i.e. *starts with the best-case assumption.* The BEP correctness can be checked automatically, by a procedure in turn viewing FSM_0 as FSM_1 or FSM_2 and identifying [11] and specifying the missing executable receptions directly in FSM_0.

Lemma 2 *[4] There is no need to worry about missing receptions for a transmission a_p after an action sequence x in FSM_p if a_p after x has previously been identified as a reception in $FSM_{p'}$.*

Upon detecting an *unexpectedly executable action sequence* in FSM_0, a designer has several options. It might turn out that the sequence is *an acceptable extension of the service.* In the opposite case, the sequence must be *prevented or its handling specified.*

An unintended reception a_p after a sequence $x_{p'}$ in $FSM_{p'}$ corresponds to a_p executed by PE_p after some $x_p \neq x_{p'}$. In other words, the two PEs initially proceed along the same path in FSM_0, but afterwards

reach a node s (a so-called *mixed node* [2]) in which they *diverge*. Suppose that upon the divergence, PE_p follows a b_p edge and $PE_{p'}$ a $c_{p'}$ edge. The divergence can be prevented by preventing concurrent execution of the conflicting actions.

The simplest way of *divergence prevention* is to assume existence of some *hidden prevention mechanism*, e.g. to require that b_p and $c_{p'}$ are two service primitives that the service users U_p and $U_{p'}$ never invoke concurrently. The solution must be reported to the procedure for identification of missing receptions, but is seldom realistic.

Usually, divergence must be prevented by *insertion of auxiliary actions*, typically internal to the server. Changing the $c_{p'}$ into $i_p c_{p'}$ would *make the choice local* to PE_p. In [1], the right to execute service actions in a formerly mixed state s of FSM_0 is initially with a PE_p. In the next step, either an a_p is executed or the right for service execution is passed to $PE_{p'}$, by an internal action i_p. In the latter case, the decision is final, i.e. [1] doesn't take care that the right is returned to PE_p if $PE_{p'}$ finds itself (or $U_{p'}$, if the service actions at p' require his/her co-operation) unable to proceed.

A more adequate procedure is given in [5], where the *PEs repeatedly exchange the execution right*, until one of them actually proceeds. The procedure can be specified in FSM_0 by splitting s into states s_1 and s_2, where each s_p inherits some of the incoming transitions of s and its outgoing transitions belonging to PE_p, and introducing a loop of internal actions (for example time-out events) (see Fig. 2). If a sufficient degree of fairness is assured, the subsequently derived service implementation will be *observationally equivalent* to the expected service.

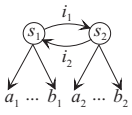

Figure 2. Specification of distributed decision-making.

Divergence prevention might be expensive. If not unacceptable for the users, it might be better to implement *recovery upon divergence*, particularly if the probability of conflicts is low. Constructing FSM_0, recovery from an unexpected sequence of service actions is specified by a designer as the service actions that should follow. The actions might represent a *new behaviour* and/or serve for *re-synchronisation* of the PEs. If the PEs diverge after executing an action sequence x, they are ready for re-synchronisation after they have extended x into such x_p and $x_{p'}$, respectively, that every message sent has also been received. Re-synchronisation can be specified simply by directing x_p and $x_{p'}$ into the same destination node [2, 7, 10].

Another kind of unexpected situations are *deadlocks*. Upon detection [11] of a potential deadlock, a tool should indicate the PE_p that should acquire an active role in the erroneous node of FSM_0. A designer should then enhance the node with an arbitrary a_p.

Whenever an a_p is introduced into FSM_0 *on purpose*, it should be marked as *desirable*, even if it serves just for recovery (though in that case it might be appropriate for a_p to be an internal action of PE_p). Upon every designer's intervention, correctness of FSM_0 must be re-checked. Hence OA makes the WFS construction process *iterative* and doesn't allow it to complete until all the executable action sequences and deadlocks are detected and handled.

OA is *user-centred*, i.e. it starts from what the users want. It is suitable if the system is being designed for a particular type of users. But even then it might happen that a designer detects that the system is capable to provide more than expected (the unexpectedly executable action sequences), and decides to declare the additional capabilities to be part of the service. Hence service synthesis should optimally combine the user-centred approach with the *system-centred* approach, i.e. the approach starting from the currently known system capabilities.

4.2 Pessimistic Approach

Following the user-centred approach, a designer is first completely and then gradually less and less optimistic, until his/her expectations drop to a realistic level that can be met by the given system. Following the system-centred approach, a designer is most interested in the level of service that the system can offer in the *worst case*, i.e. if relying solely on the information that is always available, for that is the service that can be tendered without hesitation. Only in the later stages of service construction, one gradually specifies how the occasionally available information should be employed. Hence that is the *pessimistic approach* (PA) [3].

Lemma 3 *[4] For the adopted channel type, a FSM_0 is a worst-case-assumption WFS (WWFS) iff the following rule is respected in every node s: If there are two outgoing edges specifying some actions a_1 and b_2, respectively, sequences $a_1 b_2$ and $b_2 a_1$ with a common destination must be specified in s.*

A designer may manipulate a FSM_0 as she/he likes, while the above rule, possibly applied to various nodes of the FSM automatically and concurrently, turns it into a WWFS.

On a higher level of abstraction, a WWFS reveals its structure of a meta-FSM, with meta-edges being

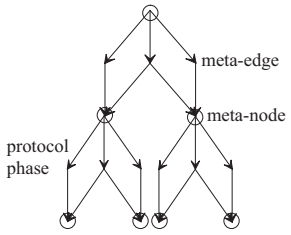

Figure 3. Identification of protocol phases in a WWFS.

clusters of sequences of ordinary edges (Fig. 3).

Lemma 4 *[3, 4] In a WWFS interpreted as a protocol specification, every meta-node with its outgoing meta-edges represents a protocol phase.*

Lemma 5 *[3, 4] In a WWFS interpreted as a protocol specification, a protocol message reception and its corresponding transmission always belong to the same meta-edge.*

In the second stage of PA, a WFS is refined to allow the PEs make *decisions based on information that is available only occasionally.*

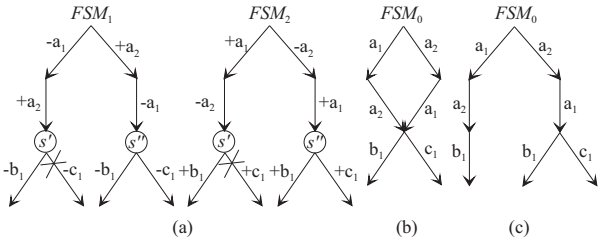

Figure 4. An example worst-case-assumption protocol and a refinement of the associated WFS.

Example: Consider the example protocol in Fig. 4(a). In the worst case, PE_1 and PE_2, respectively, execute $-a_1+a_2$ and $-a_2+a_1$, so that none of them can tell which of the two paths has been followed by the other. Consequently, PE_2 in s'' must be ready to receive both b_1 and c_1. In the best case, both PEs follow the same path, e.g. PE_2 executes $+a_1-a_2$, knowing that PE_1 has executed $-a_1+a_2$ (in the worst case, the information was not available to PE_2). Consequently, it is legal to specify that PE_2 in s' doesn't want to receive c_1, i.e. requires PE_1 not to send c_1 in s', thus c_1 is also removed from s' in FSM_1. The refinement would transform the associated WWFS (b) into the WFS in Fig. 4(c).

The first stage of OA results in a FSM_0 that might not be a WFS, hence in the second stage, the missing receptions *must* be identified and their handling specified. As the handling is semantics-dependent, its specification can not be automated. The first stage of PA results in a FSM_0 that is a WFS, hence its *further refinement is not absolutely necessary.* If desired, it can be conducted on the (sufficiently unfolded) FSM_0 simply *by selecting the edges designating actions that should be illegal at the given point of the service*, i.e. non-executable for their receiving PE. Afterwards, the corresponding transmissions can be identified and FSM_0 pruned accordingly by an automated procedure, resulting in a refined WFS.

Illegality of a transmission might imply illegality of a reception, just like illegality of a reception implies illegality of the corresponding transmissions [4]. Hence deleting a WFS edge might require unintended deletion of some other edges. As it is usually more important to keep the desirable sequences of service actions than to avoid the undesirable ones, every pruning step should be retractable. In an extreme case, a tool could be given a list of all the desirable sequences and instructed to automatically prune the WFS as long as no desirable edge is deleted or deadlock introduced. Deadlock detection can benefit from the following

Lemma 6 *[4] In the protocol specified by a FSM_0, s' and s'' in a stable state $[s', s'']$ always originate from the same state in the associated WWFS.*

As the second stage of PA doesn't introduce any new edges, every pair of a protocol message transmission and a corresponding reception is already known from the WWFS. Hence identification of such pairs can benefit from the knowledge of the meta-FSM. More generally speaking, pruning doesn't change boundaries of protocol phases, except that it might refine a phase into subphases [4]. In the case of OA, protocol phases can not be reliably established until all the possible conflicts have been identified and handled.

The final step in PA-based WFS construction is identification of the desirable edges. Initially, all edges should be marked as desirable. In the following, *an edge should be marked as undesirable exactly if its intended deletion has been found illegal, but not because it would introduce a deadlock.* The second part of the criterion reduces complexity of the subsequent protocol optimisation described in Section 5.1.

Since OA and PA are respectively based on FSM augmentation and FSM pruning, another criterion for choosing between the two could be the desired degree of concurrency in the service. If the degree is high, the final FSM_0 will probably be very similar to its approximation resulting from the first stage of PA. If the degree is low, concurrent execution of service actions will be scarcely specified, resulting in only few situations requiring conflict resolution in OA.

5 Protocol Optimisation

5.1 Reducing the Probability of Divergence

If an *undesirable action* in a WFS is an $a_{p'}$, PE_p must implement it (as a reception), for PE_p can not prevent the message's arrival. If the action is an a_p, however, PE_p should *implement it only if necessary*

for deadlock prevention. In the following, undesirability of actions shall be indicated by putting them into parentheses [7].

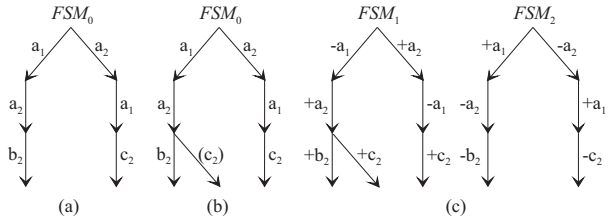

Figure 5. A WFS with an undesirable action.

Example: In Fig. 5, (a) represents the originally desired sequences of service actions and (b) the final WFS. In FSM_2, -c_2 is not necessary after +a_1-a_2, because PE_2 can always proceed with -b_2. However, the c_2 is executable in FSM_1 as a reception, and thus present in the WFS as an undesirable action.

The simplest way to indicate that an undesirable a_p in a FSM_p should be implemented is to mark it as desirable, with an additional note that PE_p should give a_p a priority lower than that of the normal actions. For every (automatically) detected deadlock state $[s', s'']$, a designer should choose an a_1 in s' or an a_2 in s'' and make it desirable, i.e. indicate the preferred direction of service continuation. The decision can in many cases be made automatically, as the following rules apply:

• If there is more than one transmission in a state s' of a FSM_p, it is for deadlock prevention irrelevant which of them is desirable. The only fact that matters is the PE_p's capability to play an active role in the state. Hence designer's intervention might be necessary only if the PE responsible can actively proceed in more than one way, of which none is yet desirable.

• If in a non-final global state there is just a particular PE_p able to play an active role, the PE responsible is automatically PE_p.

Anyhow, one could as well instruct a tool to make all the decisions on its own, for various solutions differ only in the probability of execution of individual originally undesirable service action sequences (the probability is for all the sequences in all solutions greater than zero, anyway [4]).

5.2 More Detailed Specification of Sequential Composition And Optimisation of Protocol Interactions

Let us for a while forget about the protocol and focus on the service, trying to precisely identify the service actions that the executing PE really must report to its peer. *Protocol messages basically serve for implementation of sequential composition.* In a FSM, sequential composition is specified by a node s connecting an incoming edge a to the outgoing edges. If a is an a_p and there is an outgoing edge $b_{p'}$, completion of a_p must be reported. But not always:

1) If a_p is an undesirable edge, it is not implemented by PE_p, and as such never executed, i.e. never reported.

2) If $b_{p'}$ is an undesirable edge, it is not implemented by $PE_{p'}$, and consequently doesn't have to be guarded by a message from PE_p.

3) Suppose that both edges are desirable. Let s' be the source node of a_p and s'' the destination node of $b_{p'}$. If there is also a path $b_{p'}\,a_p$ from s' to s'', with both edges desirable, it is not obvious whether a_p and $b_{p'}$ must indeed be executed strictly one after another (in either order) or their concurrent execution is also allowed (i.e. no synchronisation needed). The FSM model is not sufficiently powerful to express that (see Section 1).

Hence for some edges in FSM_0 it can be automatically established whether they require reporting, while for the (automatically identified) doubtful cases, a tool should seek designer's advice, unless instructed to automatically handle all such cases one way or the other.

In the following, let an asterisk indicate for an a_p that PE_p must report it. Likewise, let an asterisk indicate for an (a_p) that $PE_{p'}$ must implement it as a reception, because it remains executable. For unambiguous identification of such (a_p) we need

Lemma 7 *[4] In a FSM_0 that is a WFS, there must be no a_p or (a_p) that would as a (potentially receiving) edge of $FSM_{p'}$ correspond to two different (potentially transmitting) a_p in FSM_p that are both desirable but only one carries an asterisk.*

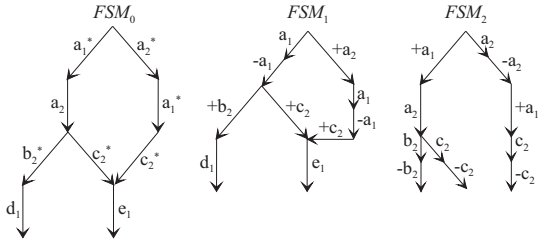

Figure 6. A protocol in which service actions are not consistently reported.

Example: The problem is illustrated in Fig. 6, where the adopted rule has been simply that the asterisk edges are implemented as receptions and the non-asterisk edges aren't. Consequently, an illegal pair of a_2 edges results in a missing +a_2 after -a_1 in the left branch of FSM_1, that would be necessary when FSM_2 diverges along its right branch.

Theorem 1 *If in a FSM_0 that is a WFS, each FSM_p has been derived from FSM_0 by transforming 1) each a_p^* into the sequence a_p-a_p, 2) each a_p left as it is,*

3) each $a_{p'}{}^*$ or $(a_{p'}{}^*)$ into $+a_{p'}$, and 4) each $a_{p'}$ or $(a_{p'})$ into a subsequently removed ε-transition, while every (a_p) or $(a_p{}^*)$ edge has been deleted, together with all the edges consequently unreachable in FSM_p, the protocol is a correct implementation of the specified service.

The WFS-construction process is very sensitive to whether two edges carry different or equal names. To avoid illegal edge pairs, it is advisable that care is taken from the beginning, that the names of the service actions immediately guarding some remote actions differ from the names of the actions without that property, unless too inconvenient. If that doesn't help, additional asterisks, i.e. additional protocol messages should be introduced automatically. For the desirable edges, we will show such asterisks in parentheses (see for example Fig. 8).

After the protocol has been completely derived, the original service action names may be restored, while preserving the associated protocol messages as they are. There is no fear that the renaming would make FSM_0 non-deterministic, for an asterisk and a non-asterisk action belonging to the same node are always semantically different from the point of the service users, so that it is inappropriate to give them the same name, anyway.

Theorem 2 *[4] If all other possibilities (explained above) to avoid illegal action pairs have been exhausted before introducing the missing asterisks automatically, the derived protocol is without redundant messages and the necessary buffer capacity of the FIFO channel from a FSM_p to $FSM_{p'}$ is the length of the longest path of consecutive receptions in $FSM_{p'}$.*

Another protocol optimisation relates to the *protocol message contents*. A message reporting a service action doesn't have to be exactly the action name. For proper service implementation, it only matters whether two messages are different or equal. Messages travelling in opposite directions are different by definition, even if their contents is the same. It is convenient that the number of different message types used within the server for each individual direction is as small as possible, for then the messages can be encoded with a small number of bits, i.e. the inter-PE traffic minimised.

Thus the final activity in protocol design should be automated renaming of protocol messages in the above spirit. The procedure could repeatedly pick a $-a_p$ and a $-b_p$ and check whether there is a state in $FSM_{p'}$ in which the corresponding $+a_p$ and $+b_p$ are both specified, but with different destination. If there is no such state, the messages can be made equal (and FSMs adequately simplified), for the renaming doesn't introduce any non-determinism.

Example: In Fig. 7, we take the example service from

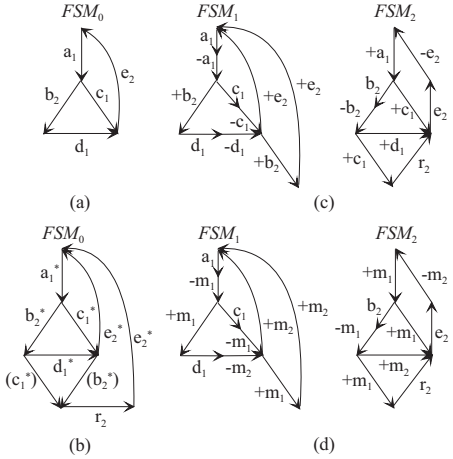

Figure 7. Protocol derivation for an example service with a mixed state.

Fig. 8 in [10], and with the help of our method obtain a protocol analogous to the one in Fig. 12 in [10]. The service specification in Fig. 7(a) with potentially conflicting b_2 and c_1 is transformed into a WFS (b) specifying that in such a conflict, c_1 is given priority, while user U_2 is informed by r_2. The derived protocol is presented in Fig. 7(c). Fig. 7(d) shows the same protocol, except that protocol messages a, b, and c have been renamed into m_1, and messages d and e into m_2, to minimise the number of the involved message types.

The renaming of protocol messages might also be crucial for proper implementation of local parallelism.

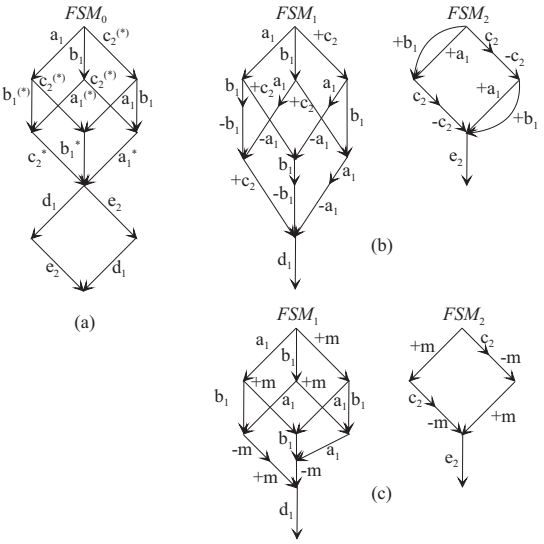

Figure 8. An example of sequential composition after concurrency.

Example: In Fig. 8, the example WFS (a) requires that a_1, b_1 and c_2 are executed in parallel, followed by parallel execution of d_1 and e_2. (b) represents the derived protocol before message renaming. From the specification it is not evident that concurrent execution of a_1 and b_1 is allowed, for the protocol message issued by PE_1 for guarding e_2 always redundantly reports also the order in

which a_1 and b_1 have been executed. By removing the redundant information, the concurrency becomes evident (c).

6 Compositional Service And Protocol Synthesis

For a complex service, it is usually possible and desirable to represent it as consisting of distinct components integrated by some typical *behaviour composition operators*. The operators easily representable in the FSM model are sequential composition, choice and iteration, but operators of any kind can be represented in the FSM model provided that

- one is satisfied with their *interleaving semantics* interpretation, i.e. it is always acceptable to see two concurrent actions as occurring one after another, in either order, and
- the possible action interleavings are describable by a *regular expression*.

[7] observes that *component integration* is typically substantially less complex if performed *on the service level*, as opposed to the previously advertised integration on the protocol level (e.g. [6]).

A composition operator on the service level might generate potential deadlocks and/or action conflicts on the protocol level. Hence besides its *basic semantics*, prescribing how the paths in the given service components should ideally be combined, each operator needs a *strategy for preventing or resolving* the above *problems arising in the particular type of distributed environment*. A tool should offer a wide range of useful composition operators, differing in one or both of the aspects.

If an operator with its basic semantics never generates action conflicts, it is appropriate that its operands are themselves WFSs. Beside the independent parallelism, typical such operator is *sequential composition*. To compose a WFS_1 with a WFS_2, a copy of WFS_2 is attached to each member of S, the set of the selected final states of WFS_1. To avoid deadlocks, there must be no pair of a $s \in S$ and a $s' \notin S$ belonging to the same meta-node of the WWFS associated with WFS_1. For observe that the two PEs might diverge to reach s and s', respectively. Hence if s has a continuation specified, so must s'. Thus the proper way for conducting the composition is to first merge all the states in S into a single state to which the PEs synchronise before entering the subsequent protocol phase. Note that for correctness of the result, it is crucial that the first operand is a WFS, for otherwise it might not represent a complete protocol phase. The same rules apply to *iteration*, for a loop is a series of sequentially composed instances of a WFS.

The proposed protocol derivation method will, where necessary, associate synchronisation messages with the *ending actions* of WFS_1 guarding the start of WFS_2. It might however be more optimal to associate messages with the *ending places* of WFS_1, i.e. with the PEs potentially executing its ending actions. For if a PE_p executes several potentially ending actions of WFS_1 concurrently, it might otherwise be that each of them generates a message, while a single, reporting completion of WFS_1 at PE_p, would suffice. Our method implements the above optimisation automatically.

Example: Consider the already discussed example in Fig. 8. We see that although PE_1 originally (Fig. 8(b)) guards e_2 by messages a_1 and b_1 induced by individual edges in the WFS, it always issues just one of them, after completion of both a_1 and b_1, as best evident from the final version of the protocol in Fig. 8(c).

On the other hand, a WFS_1 guarding a WFS_2 might consist of two alternatives with different ending places. It is appropriate that only the ending places of the alternative selected in the particular server run send synchronisation messages. Our method has no problem with the optimisation.

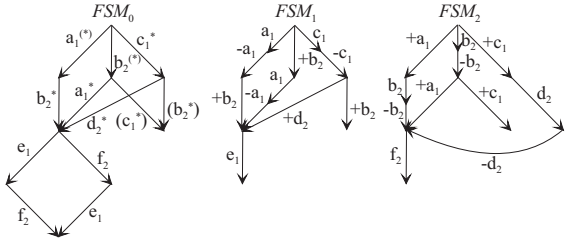

Figure 9. An example of sequential composition after alternatives.

Example: In the example in Fig. 9, the WFS initially executes two alternatives: a_1 and b_2 in parallel, or c_1 followed by d_2. If the PEs manage to select one of them consistently, the server enters parallel execution of e_1 and f_2. Unlike the first alternative, the second one has just one ending place, and only that place issues a message (d_2) guarding execution of the subsequent protocol phase.

If an operator merges a state s_1 of one server component with a state s_2 of another, that might generate a conflict. Let s be the name of the resulting state. To simplify the discussion, let's assume that the outgoing edges of s_1 are different from the outgoing edges of s_2. If the newly introduced conflicts are rather resolved than prevented, it is appropriate that both service components are themselves WFSs [7], because conflict resolution operates on individual pairs of action sequences, and the pairs newly introduced and handled on the level of s are different from those handled within s_1 and s_2 individually.

However, if we require that the newly introduced conflicts in s are handled by prevention, and the conflicts within s_1 or s_2 individually are also handled by prevention, the *conflict-prevention procedures might*

interfere. In such a case, one should first remove the edges implementing the prevention within s_1 and s_2, and afterwards introduce an integral prevention procedure for the entire s.

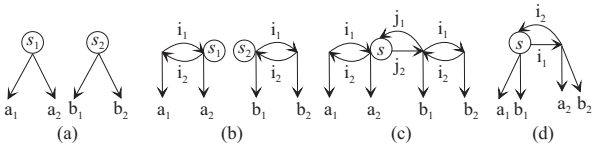

Figure 10. An example of non-optimal compositional implementation of choice.

Example: In Fig 10, (a) shows the desired actions in each of the two service components. As both components require distributed decision-making, they are in (b) enhanced with loops of internal actions (explained in Section 4). The components have different starting places, therefore their alternative composition requires another loop for distributed decision-making (c). (d) shows a more efficient, though non-compositional solution.

Obviously *it is not always the best idea to limit component integration on the service level to well-formed components*. There are cases where it is more appropriate to specify in the components just the desired service action sequences, while postponing specification of the necessary conflict handling to the resulting integrated service specification.

7 Discussion And Conclusions

With the above guidelines, it would be possible to construct a service/protocol synthesis tool that would allow a designer to concentrate entirely on service synthesis. Service synthesis would proceed interactively, with the designer specifying the desirable service action sequences, and the tool warning of the mixed states and deadlocks and automatically specifying the unexpectedly executable sequences.

For constructing such a tool, the key observation has been that for the adopted server architecture, WFS synthesis reduces to non-service-based protocol synthesis, that has already been thoroughly studied before. Having constructed a WFS with the available algorithms for protocol synthesis, it is then possible to precisely plan the necessary interactions for a protocol implementing the service in an optimal way.

Unfortunately, the proposed approach is [4] not directly applicable to non-ideal or bounded channels, to multi-party servers, or to real-time services. Nevertheless, we shall try to adapt our approach for such servers and services, because we have already successfully employed similar approaches for non-service-based protocol construction. Allowing service actions to manipulate virtual global variables would also be interesting.

Another research direction would be invention of new service composition operators, to support automatic specification of a wide range of desirable service action combinations, and also of different conflict resolution strategies. Though, once a designer is given an opportunity to construct services in a compositional way, (s)he might no longer so tightly cling to the FSM model, and willingly divert to more expressive and efficient FDTs.

8 References

[1] P.-Y. M. Chu and M. T. Liu, Synthesizing protocol specifications from service specifications in FSM model, in: *Proc. Computer Networking Symp.'88* (IEEE Comput. Soc. Press, Washington, DC, 1988) 173–182.

[2] M. G. Gouda and Y.-T. Yu, Synthesis of communicating finite-state machines with guaranteed progress, *IEEE Trans. on Communications* **32** (1984) 779–788.

[3] M. Kapus-Kolar, Constructing communication protocols on reliable bounded FIFO channels without overspecification, *Microprocessing & Microprogramming* **28** (1989) 55–58.

[4] M. Kapus-Kolar, *A Broader Perspective on Service-Based Synthesis of Two-Party Protocols*, Jožef Stefan Institute Technical Report 8191 (2000)

[5] R. Langerak, Decomposition of functionality: A correctness-preserving LOTOS transformation, in: L. Logrippo, R. Probert and H. Ural (eds.), *Protocol Specification, Testing, and Verification X* (North-Holland, Amsterdam, 1990) 229–242.

[6] H.-A. Lin and C.-L. Tarng, An improved method for constructing multiphase communication protocols, *IEEE Trans. Computers* **42** (1993) 15–26.

[7] M. Nakamura, Y. Kakuda and T. Kikuno, On constructing communication protocols from component - based service specifications, *Computer Communications* **19** (1996) 1200–1215.

[8] K. Saleh, Synthesis of communication protocols: An annotated bibliography, *Computer Communication Review* **26** (1996) no. 5, 40–59.

[9] K. Saleh and R. L. Probert, A service-based method for the synthesis of communications protocols, *International Journal of Mini and Microcomputers* **12** (1990) 97–103.

[10] K. Saleh and R. L. Probert, An extended service-based method for the synthesis of protocols, in: *Proc. 6th Bilkent Intern. Symp. on Computer and Information Sciences* (Elsevier, Amsterdam, 1991) 547–557.

[11] P. Zafiropulo, C. H. West, H. Rudin, Towards analyzing and synthesizing protocols, *IEEE Trans. on Communications* **28** (1980) 651–661.

Monika Kapus-Kolar received the BS degree in electrical engineering from the University of Maribor, Slovenia, in 1981, and the MS and PhD degrees in computer science from the University of Ljubljana, Slovenia, in 1984 and 1989, respectively. Since 1981 she has been with the Jožef Stefan Institute, Ljubljana, where she is a senior researcher at the Department of Digital Communications and Networks. Her main research interests include formal specification techniques and methods for development of distributed systems and computer networks.