# ON THE-FLY RE-SYNTHESIS OF COMMUNICATIONS PROTOCOLS

KHALED EL-FAKIH+, KASSEM SALEH* AND GREGOR VON BOCHMANN+
+University of Ottawa, School of Information Technology and Engineering
{kelfakih,bochmann}@site.uottawa.ca, Ottawa, Ontario, Canada K1N 6N5
*American University of Sharjah, Dept. of Computer Science
ksaleh@aus.ac.ae, P.O.Box 26666, Sharjah, UAE

**ABSTRACT.** Communications protocols re-synthesis involves the augmentation of existing synthesized protocol entities given a modified formal service definition. This process would reduce the time required for deploying enhanced and frequently modified services. In this paper, we introduce a re-synthesis technique based on a service-oriented protocol synthesis method introduced earlier [1]. Re-synthesis can be applied in various distributed systems application areas, such as discrete event distributed controllers, communications protocol converters, and distributed software agents.

**KEYWORDS.** Communications Protocols, Re-synthesis, Service Specification, Synthesis.

## 1. INTRODUCTION

Communications protocol synthesis involves the automatic derivation of protocol entity specifications starting from a formal communications service specification [2]. This derivation process is normally performed offline prior to the implementation and deployment of the distributed protocol entities.

At a high level of abstraction, a communication system can be viewed as a black box offering some specified communication services to a number of service users accessing the system through many distributed service access points (SAPs) using well defined Service Primitives (SPs). However, At a more refined level of abstraction, the communication services are provided using a number of cooperating protocol entities exchanging protocol messages using a reliable FIFO (first-in, first-out) communication medium. A protocol entity uses the lower (underlying) service functions to relay protocol messages to other protocol entities. The N-Layer protocol design process starts from a complete service specification, and is formulated as "the design of (N-)protocol specifications starting from both (N-) and (N-1) Service Specifications".

The service is described by a FSM which specifies the legal sequences of SP occurrences that can be observed at the distributed SAPs. A service specification, S-SPEC, is formally defined by a tuple (Ss, Σs, Ts, σ), where: Ss is a non-empty finite set of service states, Σs is a finite set of service primitives, Ts is a partial transition function between service states, and $\sigma \in Ss$ is the initial service state. A service primitive $SP \in \Sigma s$ identifies the type of service event and the SAP(s) at which it may occur.

For every node representing a service state $s \in Ss$, *OUT*(*s*) denotes the set of SAPs associated with the SPs of its outgoing edges, and INS$\downarrow$(s) denotes the set of incoming edges to s, each of type $\downarrow$. Moreover, for an edge E of an FSM, let *s_source* and *s_dest* be source and destination states of E, respectively. For each incoming edge E, PRE(E) denotes the source state of E. For an edge E$\downarrow$ (E$\uparrow$) of S-SPEC, PEi(E) denotes the protocol entity that receives (sends) the message associated with E.

A service primitive Si is of type $\uparrow$, written Si$\uparrow$, if the SP is directed upward from the protocol entity PE-SPECi to SAPi. Similarly, Si is of type $\downarrow$, written Si$\downarrow$, if the SP is directed downward from the service user at SAPi to the protocol entity PE-SPECi.

The projection onto a set X of SAPs (Πx) is a unary function which can be applied to a finite state machine (FSM) (S, Σ, T, σ) yielding another FSM (S, Σ', T', σ) in which Σ' is a subset of $\Sigma \cup \{\varepsilon\}$ and T' = T with relabeling to ε of events in Σ not contributing to the SAPs onto which the FSM is projected.

A projected S-SPECi (PS-SPECi) is the projection of the (FSM) service specification S-SPEC onto SAPi (PS-SPECi = ΠSAPi S-SPEC). PS-SPECi is

represented by (Ss, Σ's, T's, σ), where Σ's, is a subset of Σs and T's is a subset of the cartesian product $Ss \times (\Sigma's \cup \{\varepsilon\}) \times Ss$.

A protocol specification (P-SPEC) consists of several interacting protocol entities (PE-SPECs). A protocol entity specification PE-SPEC is formally defined by a tuple (Sp, Σp, Tp, σ), where: Sp is a non-empty finite set of protocol states, Σp is a finite set of protocol events, $\Sigma p = \Sigma's \cup IPE$, where $\Sigma's \subseteq \Sigma s$, and IPE is the set of internal protocol events, Tp is a partial transition function between protocol states, and $\sigma \in Sp$ is the initial protocol state.

Our re-synthesis is service-based, that is, it considers requirement changes at the service definition, in contrast to the work by Bista et al. [4], which consider changes at the protocol level, i.e., in one of the protocol entities.

In this paper, we consider the re-synthesis problem as the systematic modification of PE-SPECs after applying a modification to S-SPEC to become S-SPEC'. The rest of the paper is organized as follows. Section 2 provides an overview of the basic protocol synthesis method. Section 3 introduces our protocol re-synthesis method and its rules along with some application examples. Finally, in Section 4, we provide some concluding remarks and future work.

## 2. PROTOCOL SYNTHESIS

In this section, we provide a brief overview of the protocol synthesis technique that we consider as the basis for our re-synthesis approach. In a PS-SPECi, two types of transitions exist: i) ε-transition which corresponds to an SP-labelled transition in a PS-SPECj ($j \neq i$), and ii) transition labelled by an SP observable at SAPi.

The core of the synthesis technique consists of the "Transition Synthesis Rules (TSRs)". These rules are applied correspondingly to each of the transitions (ε- or SP-labelled) in PS-SPECi. The intuitions for these rules are given below.

*a.Transition labelled by an SP E in PS-SPECi:*

Rule a.1: This rule implies that the flow of control needs not be transferred to another protocol entity (or service user), and therefore a synchronization message should not be transmitted at this point.

Rule a.2: Since the SP is originating from the service user, and is taking the service back to its initial state, synchronization messages must be sent to all other protocol entities to synchronize at the initial state in each of the respective protocol entities. This would synchronize the protocol at the same initial global stable state.

Rule a.3: In this case, the SP is originating from the protocol entity and is taking the service back to its initial state. However this SP is most probably a result of a reset protocol message, and therefore there is no need to transmit any other protocol message.

Rule a.4: In this case, the SP is originating from the service user at SAPi. Following the occurrence of this SP, other SPs can be observed at other SAPs, therefore a synchronization message should transfer the flow of control to other corresponding protocol entities.

Rule a.5: The intuition for this rule is similar to Rule a.3.

*b. Transition labelled by ε in PE-SPECi:*

In Rules b.1, b.3 and b.5, PE-SPECi must not expect any message at this point, since according to the service specification, no service action is expected at SAPi.

Rules b.2 and b.4 complement Rules a.2 and a.4. Reception transitions are synthesized and correspond to the message transmissions synthesized in Rules a.2 and a.4.

The synthesis process starts from an FSM specification of the service (S-SPEC), and automatically derives the protocol entities that provide the set of services given in S-SPEC.

Steps of the synthesis algorithm:

1. Project the service specification S-SPEC onto each SAP to obtain the PS-SPECs.
2. Apply a TSR to each transition in the PS-SPECs to obtain PE-SPECs.
3. Using the algorithms described in [3], remove ε-cycles and ε-transitions to obtain the PE-SPECs as reduced and equivalent finite state machines.

## 3. PROTOCOL RE-SYNTHESIS

In our proposed re-synthesis technique, we are assuming that modifications to the service specification from S-SPEC to S-SPEC' will not invalidate the basic FSM model assumptions. We also assume that we apply the re-synthesis to the derived PEs. In this case, we might have to merge/split states during the re-synthesis process, thus we have to know if there exists more than one ε-path(s) between two states. This can simply be achieved using a linear-time algorithm we developed. The operations that we consider on S-SPEC are: adding an edge, and removing an edge, adding/Removing a state (or a node) with its incident edges.

### 3.1. ADDING AN EDGE TO S-SPEC

I.1) Adding a new edge, s_source —↓$E_i$→ s_dest., between states s_source and s_dest, where s_dest is not the initial state of S-SPEC.

(i) For PEi(E):
-Add the edge labeled E/!(e, [OUT(s_dest) - SAPi]) between state(s) s_source and s_dest of E, where SAPi = PEi of E. This corresponds to Synthesis Rule a.4.

(ii) ∀PEj, j≠i and j ∈ OUT(s_dest):
-Do exactly the same as (i) above, but label the new edge as ?e. This corresponds to Synthesis Rule b.4.

(iii) For PEj, j≠i and j ∉ OUT(s_dest.):
-Add ε-edge between state(s) s_source and s_dest.
-Apply ε-removal algorithm.

(iv) If OUT'(s_source) – OUT(s_source) = PEi(E)
(a)- For each t ∈ INS↓ (s_source), where s_source is not the initial state, and
If PEi(E) ≠ PEi(t):
-In PEi(t) Change the label of the transition:
PRE(t) — A/!a(OUT(s_source) - SAPi of t) → s_source to PRE(t)— A/!a(OUT'(s_source)- SAPi of t)→ s_source
This corresponds to Synthesis Rule a.4.

(b)- If PEi(E) ≠ PEi(t), for PEi(E) do: /*Note: PEi(E) = OUT'(s_source) – OUT(s_source) */
1)- Remove an ε-edge between states PRE(t) and s_source in PEi(E). This reverses the effects of Synthesis Rule b.5.
2)- Add the edge ?a between the states PRE(t) and s_source in PEi(E), where a is the label/msg of edge t. This corresponds to Synthesis Rule b.4.

I.2) Adding a new edge, s_source —↓$E_i$→ s_dest., between states s_source and s_dest, where s_dest is the initial state of S-SPEC.

(i) For PEi(E):
- Add the edge labeled E/!(e, [All SAPs- SAPi]) between state(s) s_source and s_dest, where SAPi = PEi of E. This corresponds to Synthesis Rule a.4.

(ii) ∀PEj, j≠i:
- Add the label ?e type edge between state(s) s_source, and s_dest. This corresponds to Synthesis Rule b.4.

I.3) Adding a new edge, s_source —↑$E_i$→ s_dest, between states s_source and s_dest. (Note here s_dest could also be the initial state)
(i) For PEi(E).
-Add the edge labeled E between state(s) s_source and s_dest. This corresponds to Synthesis Rule a.3.

(ii) ∀PEj, j≠i:
- Add an ε-edge between state(s) s_source, and s_dest. This corresponds to Synthesis Rule b.3.
- Apply the ε-removal algorithm.

(iii) If OUT'(s_source) – OUT(s_source) = PEi(E)
- Do exactly as specified in I.1.iv.

#### 3.1.1 ADDING EDGES TO S-SPEC

In this section, we consider the service specification FSM, S-SPEC in Figure 1, and its synthesized protocol entities PE1, PE2 and PE3, in Figure 2. Then, we consider the application of independent service modifications. For each modification, we apply the appropriate re-synthesis rules to modify the synthesized protocol entities, hence obtaining, PE'1, PE'2, and PE'3.

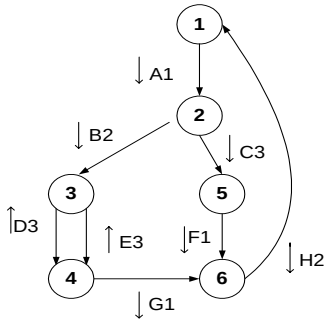


**Figure 1.** S-SPEC.

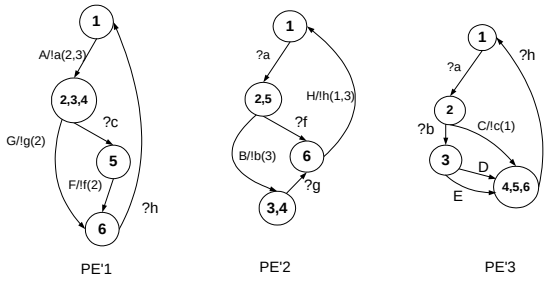


**Figure 2.** Synthesized PE-SPECs of S-SPEC of Figure 1.

Operations on S-SPEC of Fig.1 , are reflected as necessary on the derived PE, yielding the re-synthesized protocol entities PE's, after applying "Adding an Edge" re-synthesis rules and algorithms of Section 3.1.I.1.

*Example 1.* Add [1] —↓I1→ [2] (transition labeled ↓ I1 from service state 1 to 2), producing S-SPEC'1.

- PE(E = I1) is PE1,
- By (i): Add to PE1: [1] — I/!I(2,3)→ [2,3,4]

- By (ii): Add to PE2: [1] —— ?i→ [2,5], and to PE3 Add : [1] —— ?i → [2]

*Example 2.* Add [4] —— ↓J2 → [6] (transition labeled ↓ J2 from service state 4 to 6), producing S-SPEC'2.

- PE(E = J2) is PE2,
- By (i): Add to PE2 : [3,4] —— J→ [6]
- By (iii): Add to PE1: [2,3,4] ——ε→[6], and to PE3: [4,5,6] ——ε→[4,5,6]. Applying the ε removal algorithm to PE1 yields:

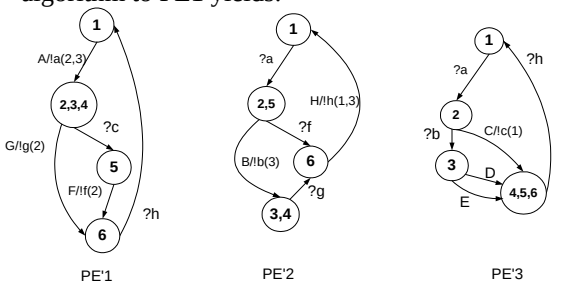


**Figure 3.** Re-Synthesized PE-SPECs after adding [4] —— ↓J2 → [6] to S-SPEC

*Example 3.* Add [3] —— ↓K2 → [4] (transition labeled ↓ K2 from service state 3 to 4), producing S-SPEC'3.

- PE(E = K2) is PE2,
- By (i): Add to PE2: [3,4] —— K/!k(1) →[3,4]
- By (ii): Add to PE1: [2,3,4] —— ?k →[2,3,4]
- By (iii): Add to PE3: [3] —— ε → [4,5,6].
- Applying the ε removal algorithm to PE3 yields:

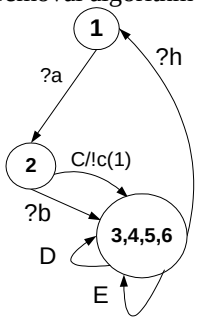


**Figure 4.** Re-Synthesized PE-SPEC PE'3 after adding [3] —— ↓K2 → [4] to S-SPEC.

*Example 4.* Add [3] —— ↓L1 → [4] (transition labeled ↓ L1 from service state 3 to 4), producing S-SPEC'4.

-PE(L1) = PE1,

By (i) : Add to PE1: [2,3,4] —— L →[2,3,4]

By (iii): Add to PE2 : [3,4] —— ε →[3,4] and Add toPE3: [3] —— ε →[4,5,6]

By (iv-a) : t = ↓B2 and PE(t) = PE2,

In PE2 change the label of the edge [2,5] —— B/!b(3) → [3,4] to [2,5] —— B/!b(**1**,3) → [3,4]

By (iv-b): For PE(L1)=PE1, split the state [2,3,4], and add the edge [2]——?b→ [3,4].

Figure 5 below, shows PE'1 after applying the above corresponding rules.

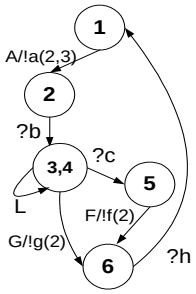


**Figure 5.** Re-Synthesized PE-SPEC PE'1 after adding [3] —— ↓L1→ [4] to S-SPEC

*Example* 5. Add [3] ——↑M3 → [4] (transition labeled ↓M3 from service state 3 to 4), producing S-SPEC'5.

- PE(E = M3) is PE3
- By (i): Add to PE3: [3] —— M→ [4,5,6]
- By (ii): Add to PE1: [2,3,4]—— ε→ [2,3,4], and toPE2 Add: [3,4] —— ε → [3,4]

*Example 6.* Add [5] ——↑O2→ [6] (transition labeled ↑ O2 from service state 5 to 6), producing S-SPEC'6.

- PE(E=O2) is PE2
- By (i): Add to PE2: [2,5] —— O→ [6]
- By (ii): Add to PE: [5] —— ε→ [6], and to PE3: [4,5,6] —— ε→ [4,5,6]
- By (iv-a) PE(t=C3)=PE3. In PE3, change the label of transition [2]——C/!c(1)→[4,5,6] to [2]—— C/!c(1,2)→ [4,5,6]
- By (iv-b): In PE2, Split the state [2,5], and Add the edge [2] ——?c→ [5]

Figure 6 shows the re-synthesized PE-SPEC's PE'1, PE'2, and PE'3 after applying the above modifications.

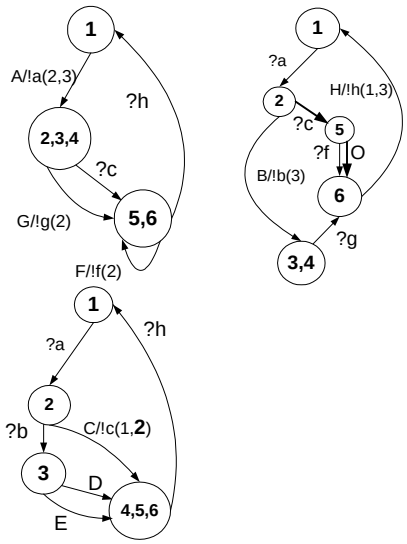


**Figure 6.** Re-Synthesized PE-SPECs after adding [5] —— ↑O2 → [6] to S-SPEC

### 3.2. RULES FOR REMOVING AN EDGE FROM S-SPEC

II.1) Removing an edge, $\downarrow$E, between s_source and s_dest, where s_dest is not the initial state of S-SPEC.

(i) For PEi(E):
-Remove the edge labeled E/!(e,OUT(s_dest.) - SAPi of E). This reverses the effects of Synthesis Rule a.4 applied to E in PEi(E).

(ii) $\forall$PEj, j$\neq$i and j $\in$ OUT(s_dest.):
-Remove the edge labeled ?e. This reverses the effectsof Synthesis Rule b.2.

(iii) $\forall$PEj, j$\neq$i and j $\notin$ OUT(s_dest.) :
-Remove an $\varepsilon$-edge between states s_source and s_dest in PEj. This reverses the effects of Synthesis Rule b.1.

(iv) If OUT'(s_source) – OUT(s_source) = PEi(E)
(a) Do exactly as specified in I.1.iv.a,
(b) If PEi(E) $\neq$ PEi(t), To PEi(E) do: /* note: PEi(E)=OUT'(s_source) – OUT(s_source)*/
(1) Re-label the transition labeled ?a between states PRE(t) and s_source in PEi(E) by $\varepsilon$, conforming to Synthesis Rule b.4.
(2) Apply the $\varepsilon$-removal algorithm to this relabeled $\varepsilon$ transition edge.

II.2) Removing an edge $\downarrow$E between s_source and s_dest, where s_dest is the initial state of S-SPEC.
(i) Do exactly as specified in as II.1.i.
(ii) $\forall$PEj, j$\neq$i: Remove the edge labeled ?e.
(iii) Do exactly as specified in II.1.iv.

II.3) Removing an edge, $\uparrow$E, between s_source and s_dest, where s_dest may or may not be the initial state of S-SPEC.
(i) For PEi(E), remove E.
(ii) $\forall$PEj, j$\neq$i:
- Remove an $\varepsilon$-edge between states s_source and s_dest in PEj. This reverses the effects of Synthesis Rule b.5.
(iii) Do exactly as specified in II.1.iv.

#### 3.2.1 EXAMPLES OF REMOVING EDGES

Operations on S-SPECs, are reflected as necessary on the derived PE, yielding the re-synthesized protocol entities PE's, after applying "Removing an Edge" re-synthesis rules and algorithms of Section 3.2.II.1.

*Example 7.* Remove [1] —$\downarrow$I1$\rightarrow$ [2] (transition labeled $\downarrow$I1 from service state 1 to 2) from S-SPEC'1 of Example-1 of Section 3.1.1.
- PE(E = I1) is PE1,
- By (i): Remove from PE1: [1] — I/!I(2,3)$\rightarrow$ [2,3,4]
- By (ii): Remove from PE2: [1]—?i$\rightarrow$ [2,5], and remove from PE3: [1] — ?i$\rightarrow$ [2]

*Example 8.* Remove [4] — $\downarrow$J2 $\rightarrow$ [6] (transition labeled $\downarrow$J2 from service state 4 to 6) from S-SPEC'2 of Example 2 of Section 3.1.1.
- PE(E = J2) is PE2,
- By (i) : Remove From PE2 : [3,4] — J $\rightarrow$ [6]
- By (iii): Remove from PE1: [2,3,4] — $\varepsilon$ $\rightarrow$[6], and from PE3: [4,5,6] — $\varepsilon$ $\rightarrow$[4,5,6].

PE-SPEC's produced after applying the above rules are shown in Figure 2.

*Example 9.* Remove [5] — $\uparrow$O2 $\rightarrow$ [6] (transition labeled $\uparrow$O2 from service state 5 to 6) from S-SPEC'6 of example-6 of Section 3.2.1.
- PE(E=O2) is PE2
- By (i) : Remove from PE2 : [5] — O $\rightarrow$ [6]
- By (ii): Remove from PE1: [5] — $\varepsilon$ $\rightarrow$ [6], and from PE3: [4,5,6] — $\varepsilon\rightarrow$ [4,5,6]
- By (iv-a) PE(t=C3)=PE3. In PE3, change the label of transition [2]—C/!c(1,2)$\rightarrow$[4,5,6] to [2]—C/!c(1)$\rightarrow$ [4,5,6]
- By (iv-b): In PE2, change the label edge [2] —?c $\rightarrow$ [5] to [2] —$\varepsilon\rightarrow$ [5], then apply $\varepsilon$ removal algorithm.

PE-SPEC's produced after applying the above rules is shown in Figure 2.

### 3.3 RULES FOR ADDING A STATE TO S-SPEC

After adding a state *s_new* and its corresponding incoming and outgoing edges to S-SPEC, producing S-SPEC' the new re-synthesized PE-SPEC' are produced as follows:
- Add a new state to each PE-SPEC
- For each outgoing edge of s_new of S-SPEC, apply "Adding an edge" re-synthesis rules of Section 3.2.1
- For each incoming edge of s_new of S-SPEC, apply "Adding an edge" re-synthesis rules of Section 3.2.1

#### 3.3.1 EXAMPLE OF ADDING A STATE

Let S-SPEC' shown in Figure 7 below be that of Fig. 1 after adding state [7] and its corresponding edges. We note that the PE-SPECs that correspond to S-SPEC of Fig.1, is shown in Fig. 2.

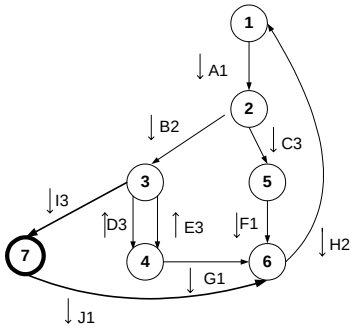


**Figure 7.** S-SPEC'

First, for the outgoing edge [7] — ↓J1 → [6] of S-PEC', we apply "Adding an Edge" re-synthesis rules of Section 3.1.I.1 to PE-SPECs of Figure 2, as follows:

- PE(E=J1) is PE1
- By (i): Add to PE1 : [7] — J/!j(2) → [6]
- By (ii): Add to PE2 : [7] — ?j→ [6]
- By (iii): Add to PE3: [7] — ε → [4,5,6]

Second, for the incoming edge [3] — ↓I3 → [7] of S-SPEC', we apply "Adding an Edge" re-synthesis rules of Section 3.1.I.1 to PE-SPECs of Figure 2, as follows:

- PE(E=I3) is PE3
- By (i): Add to PE3 : [3] — I/!i(1) → [7]
- By (ii): Add to PE1 : [2,3,4] — ?i → [7]
- By (iii): Add to PE2: [3] — ε → [7]

After applying adding a state re-synthesis rules, we get the following re-synthesized PE-SPECs.

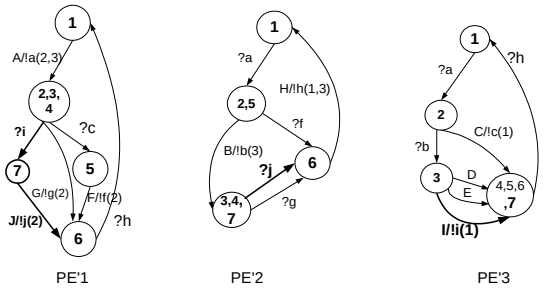


**Figure 8.** Re-Synthesized PE-SPECs of S-SPEC' of Figure 7.

### 3.4 RULES FOR REMOVING A STATE

After removing a state s_remove and its corresponding incoming and outgoing edges from S-SPEC, producing S-SPEC' the new re-synthesized PE-SPEC' are produced as follows:

- For each outgoing edge of s_remove of S-SPEC, apply "Remove an edge" re-synthesis rules of Section 3.2.1
- For each incoming edge of s_remove of S-SPEC, apply "Remove an edge" re-synthesis rules of Section 3.2.1
- Remove the states that corresponds to s_remove from each PE-SPEC

## 4. CONCLUSIONS

In this paper, we have introduced a protocol re-synthesis method which modifies existing protocol entities specifications after modifications to the service specifications are made. This approach allows the faster deployment of modified protocol entities instead of applying an initial synthesis process to the service specification. This re-synthesis technique can have applications in many areas where small changes to the provided service are introduced very frequently. As a result, the modified service can be introduced more efficiently and quickly. In the future, we intend to prove the correctness of the re-synthesis rules, and apply the method to specific application areas, such as telephony, distributed supervisory control, and distributed databases. Moreover, we have extended the method to systems modeled as Extended Petri Nets.

## ACKNOWLEDGEMENT

The authors would like to acknowledge the support of this work by Communications and Information Technology Ontario (CITO) and by the American University of Sharjah.

### REFERENCES

[1] K. Saleh and R. Probert, "Automatic synthesis of protocol specifications from service specifications", *Inter. Phoenix Conf. On Computers and Communications (IPCCC'1991)*, Phoenix, Arizona, USA, pp. 615-621.

[2] R. Probert and K. Saleh, "Synthesis of communications protocols: Survey and Assessment", *IEEE Trans. on Computers*, Vol. 40, pp. 468-476, April 1991.

[3] A. Khoumsi and K. Saleh, "Two formal methods for the synthesis of discrete event systems", *Computer Networks and ISDN Systems*, Vol. 29, No. 7, pp. 759-780, July 1997.

[4] B.B. Bista, K. Takahashi, H. Kaminaga, and N. Shiratori, "A flexible protocol synthesis method for adopting requirement changes", Proc. of the *1996 Intern. Conf. On Parallel and Distributed Systems*, Tokyo, Japan, pp. 319-326 June 1996.