Al Dallal J, Saleh KA. Synthesizing distributed protocol specifications from a UML state machine modeled service specification. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 27(6): 1150–1168 Nov. 2012. DOI 10.1007/s11390-012-1293-1

Synthesizing Distributed Protocol Specifications from a UML State Machine Modeled Service Specification

Jehad Al Dallal and Kassem A. Saleh

Department of Information Science, College for Women, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait

E-mail: j.aldallal@ku.edu.kw; saleh.kassem@yahoo.com

Received October 31, 2011; revised September 4, 2012.

Abstract The object-oriented paradigm is widely applied in designing and implementing communication systems. Unified Modeling Language (UML) is a standard language used to model the design of object-oriented systems. A protocol state machine is a UML adopted diagram that is widely used in designing communication protocols. It has two key attractive advantages over traditional finite state machines: modeling concurrency and modeling nested hierarchical states. In a distributed communication system, each entity of the system has its own protocol that defines when and how the entity exchanges messages with other communicating entities in the system. The order of the exchanged messages must conform to the overall service specifications of the system. In object-oriented systems, both the service and the protocol specifications are modeled in UML protocol state machines. Protocol specification synthesis methods have to be applied to automatically derive the protocol specification from the service specification. Otherwise, a time-consuming process of design, analysis, and error detection and correction has to be applied iteratively until the design of the protocol becomes error-free and consistent with the service specification. Several synthesis methods are proposed in the literature for models other than UML protocol state machines, and therefore, because of the unique features of the protocol state machines, these methods are inapplicable to services modeled in UML protocol state machines. In this paper, we propose a synthesis method that automatically synthesizes the protocol specification of distributed protocol entities from the service specification, given that both types of specifications are modeled in UML protocol state machines. Our method is based on the latest UML version (UML2.3), and it is proven to synthesize protocol specifications that are syntactically and semantically correct. As an example application, the synthesis method is used to derive the protocol specification of the H.323 standard used in Internet calls.

Keywords distributed system, network protocol, object-oriented design method

1 Introduction

Reliable computer communication protocols play a critical role in providing effective communication services. A communication protocol consists of a set of rules that govern the orderly exchange of information among network components to provide a specified set of services to service users located at different access points that are typically geographically distributed. A full protocol definition defines a precise format for valid messages (syntax), rules for the data exchange (grammar), and a vocabulary of valid messages that can be exchanged, with well-defined meanings (semantics). The communication service specification describes the distributed functions that a communication system must provide to its service users. The specification of a communication protocol includes the specification of the communicating protocol entities, each servicing a particular service access point.

Fig.1 demonstrates the relation between the service specification and the protocol specification. At a high level of abstraction, a communication system can be viewed as a service provider that offers some services to users who access the system through many distributed service access points (SAPs), using service functions called service primitives (SPs) (Fig.1(a)). The SP identifies the type of the event and the SAP at which it occurs. The specification of the service provided by the layer is defined by the ordering of the visible SPs and is called the service specification (S-SPEC). At a more refined level of abstraction, the communication services are provided to the service users by a number of cooperating protocol entities that exchange protocol messages through a communication medium (Fig.1(b)). The protocol specification (P-SPEC) describes the exchange of messages between the protocol entities.

Regular Paper
This work is supported by Kuwait University under Research Grant No. WI03/10.

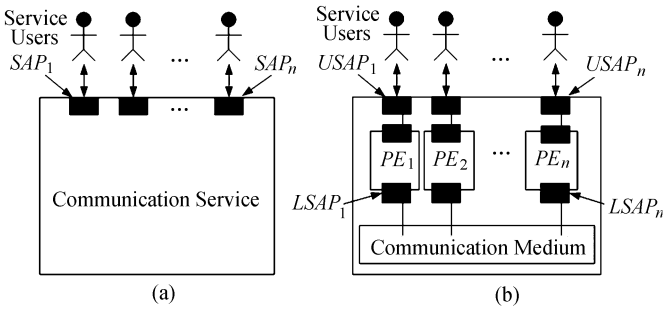

Fig.1. Communication service and protocol concepts. (a) Communication service. (b) Communication protocol. USAP: upper SAP, LSAP: lower SAP, PE: protocol entity.

The protocol design includes constructing interacting protocol entities to provide a set of services specified in the service specification. Because of their complex nature, communication protocols are difficult to design properly in an informal and non-methodical fashion[1]. Design errors will eventually result in an erroneous protocol implementation unless they are detected at an early stage. There are two types of design errors: semantic and syntactic. Semantic design errors cause the provision of incorrect or incomplete services to distributed protocol users, which implies that the designed protocol is not consistent with the service specification. Syntactic design errors can cause the protocol to deadlock. To verify the correctness of the protocol specification, a time-consuming sequence of design, analysis, error detection and correction has to be applied iteratively until the design of protocol specifications of the distributed entities becomes error-free.

To overcome this problem, the protocol design process can start from a complete and unambiguous service specification. The construction of a protocol specification from a given service specification is called *protocol synthesis*. If the applied synthesis method is proven to derive a correct and error-free protocol specification, the resulting protocol specification will not require any further validation. The synthesis approach is used to construct or complete a partially specified protocol design such that the interactions between the constructed or completed protocol entities proceed without encountering any logical error and provide the specified service. In addition, the syntactic and semantic correctness of the synthesized protocol is often a direct byproduct of the synthesis method[2]. Several protocol synthesis methods have appeared in the literature (e.g., [3-12]), where both S-SPEC and P-SPEC are modeled in Petri nets, finite state machines (FSMs), or other formal languages such as LOTOS. These methods are applied to synthesize and design the required protocol specification. Once the design is complete, a programming language is used to implement the required protocol specification.

Because of the promised advantages of the object-oriented approach, such as high reusability, modularity, and maintainability[13], over the past decade, developers have shifted towards using object-oriented programming languages such as Java and C++. As a result, object-oriented programming has been widely used in the software industry for various applications such as the development of communication systems (e.g., [14-20]).

To develop an object-oriented system, developers use the Unified Modeling Language (UML), a standardized, general-purpose modeling language used to specify, visualize, modify, construct, and document the artifacts of an object-oriented application[21]. UML is the most widespread software modeling language. It can be applied to the design modeling of any type of software project, and it is the standard language adopted by industry for modeling the design of object-oriented systems[22]. The UML state machine (previously referenced as a statechart) has been widely used to describe the service and protocol specifications for communication systems (e.g., [23-33]). Despite the popularity of using an object-oriented paradigm in developing modern communication systems, as far as we know, none of the existing synthesis methods is applicable for UML state machines. The existing synthesis methods construct the protocol specifications from the service specifications described by conventional models such as Petri nets and finite state machines. As a result, because of the absence of the UML-based synthesis methods, instead of automatically constructing the proven error-free protocol specification directly from the service specification, current developers of object-oriented communication systems are forced to use traditional design methods, i.e., applying the sequence of design, analysis, error detection, and error correction on the designed protocol specification iteratively until the design becomes error-free.

UML is supported by many available tools (e.g., [22, 34-40]). These tools support editing of the UML diagrams and fully- or semi-automating the generation of the corresponding object-oriented code. This support will be missed if other modeling diagrams such as Petri nets and conventional finite state machines are used. In addition, UML state machines are capable of modeling specifications that include nested hierarchical states. This feature reduces the complexity of the state machine diagram and allows for the building of more expressive diagrams[30]. Allowing for nested hierarchical states is a feature that is neither supported by Petri nets nor by conventional finite state machines, which

makes the synthesis methods based on these two models inapplicable for service specifications that include hierarchical states. Autonomous communication systems, such as the Internet and mobile communication systems, are currently widely used (e.g., [24, 29, 32]). In such systems, a user can initiate a service at any time. As a result, distributed users at different SAPs may issue simultaneous service primitives, and consequently, it is possible that two or more users will simultaneously issue service requests to each other. This situation leads to message collision and requires careful consideration during protocol design. Unfortunately, conventional finite machines do not model concurrency behaviors, and therefore, their corresponding synthesis methods are inapplicable to concurrent systems that can be modeled in UML.

In this paper, we propose a new synthesis method that automatically derives protocol specifications from a service specification, where both the service and the protocol specifications are modeled by UML state machines. Our method is based on the latest UML version[21]. The synthesis method considers complex features of state machines that include concurrency and hierarchical organization. The resulting protocol specifications are guaranteed to conform to the service specification and to be free of syntactic errors. Therefore, once the service specification is synthesized, the resulting protocol specification does not require any further verification and will be ready to be fed to the code generation tools (e.g., [34, 36, 38, 41]) to automatically or semi-automatically implement the resulting protocol specification. This paper provides an analysis for the space and time complexity of the proposed synthesis algorithm and shows one of its real applications. This application considers the call establishment service of the H.323 standard used in Internet calls.

This paper is organized as follows. In Section 2, we present an overview of related research. The models used for the service and protocol specifications are defined in Section 3. In Sections 4, 5, and 6, the UML-based protocol synthesis method is introduced, its space and time complexity is analyzed, and its correctness is proven. An application for the proposed synthesis method is illustrated in Section 7. Finally, Section 8 provides conclusions and a discussion of future work.

2 Related Work

This section provides an overview of the existing protocol synthesis methods and provides a brief description of the UML state machine diagram.

2.1 Overview of Synthesis Methods

A substantial amount of research has been performed on the development of formal methods for the design of communication protocols. These methods follow one of two types of design approaches, namely, the analysis or synthesis approach[2]. In the analysis approach, the protocol designer starts with a preliminary version of the protocol in which the syntactic and semantic validation aspects are often overlooked. The preliminary version is usually obtained by defining messages and the effect of message exchanges on the protocol entities under design. This approach often results in an incomplete and erroneous design. Therefore, a design verification and analysis process is then performed to detect errors and omissions in the protocol design. The process of re-design, analysis, error detection and correction is applied iteratively until the protocol design becomes error-free. This approach is time consuming because of its iterative, trial-and-error nature. Design validation and analysis techniques have been proposed by several researchers (e.g., [1, 31, 42-45]). In the synthesis approach, the protocol design is constructed or completed in such a way that no further validation is needed. Some protocol synthesis methods initiate the derivation process from a complete service specification (e.g., [2, 6-9, 11, 47-48]), and others do not (e.g., [5, 8, 49]). The protocol synthesis methods can be further classified according to the employed models. The models that are used include finite state machines (e.g., [2, 7, 9, 11-12]), Petri nets (e.g., [7, 47-48]), and LOTOS-like (e.g., [6, 8]).

UML has been shown to be useful in modeling communication protocols (e.g., [27-28, 32]). Specifically, UML state machines are widely used in modeling protocols for object-oriented communication systems (e.g., [23-32]). However, as per our knowledge, no protocol synthesis method is available to derive the protocol specification starting from a service specification modeled in UML state machines. The existing synthesis methods for other types of models cannot be applied to UML state machines because these methods do not consider hierarchical state machines. Moreover, some of the synthesis methods, such as those based on FSM models, are not capable of dealing with the concurrency behaviors of distributed systems.

2.2 State Machine Diagrams: Concurrency and Hierarchical Representations

In UML 2.x[50], a state machine (previously referenced as a statechart in UML 1) is a diagram where typical states are shown in rounded-corner rectangles connected with labeled arrows that represent transitions. There are two types of state machines: the behavior state machine, which describes the behavior of a

part of a system, and the protocol state machine, which describes the usage protocol of a part of a system. A protocol state machine expresses legal transitions and their order. In this paper, we are concerned with the latter state machine type because it can be used to describe communication protocols. For the rest of this paper, we will simply use the term "state machine" instead of "protocol state machine".

In a state machine, a transition is an allowable two-state sequence. Each transition can be associated with 1) an event, 2) a set of predicates, and 3) a set of expected actions. To execute a transition, the protocol must be in the accepting state of the transition, the event is executed, and the predicates are evaluated to true. The UML syntax for a transition is:

event-name [guard predicate]/action-expression.

A state can be simple, composite, or a submachine. A simple state is a state that does not have any substates. A composite state may contain states. These states are called substates and can be of any of the three types of states, which form nested or hierarchical states. A submachine state is semantically equivalent to the composite state, and it represents another state machine. A composite state can include one or more orthogonal regions. The regions are separated by dashed lines to represent a concurrent behavior. Each region includes substates connected by transitions. The states and transitions in different regions are orthogonal, and they are concurrently executed.

From a different point of view, a state can be classified as typical or special. A typical state expresses a stable situation that represents the state context. Special states have different semantics, and they include initial, final, join, fork, junction, and choice states. Each of the initial, junction, and choice states is represented by a solid filled circle. An initial state of a state machine represents the starting state of the protocol represented by the state machine. An initial state of a region in a composite state represents the starting state of the protocol specified by the region. An initial state has an unlabeled outgoing transition. In this paper, we refer to the destination state of this unlabeled outgoing transition as a *stable initial state*. A junction state is used to attach its incoming transitions together. A choice state is used to attach outgoing transitions together, and when this state is reached, the executed transition is the transition that has its guard predicates satisfied. A final state is represented by a circle surrounding a small, solid-filled circle, and it expresses the completion of the protocol specified in a composite state region or described by the state machine.

A state machine must have a single initial state and can have multiple final states. A region in a composite state can have at most one initial state and can include multiple final states. An incoming transition to a composite state represents an incoming transition to the initial state in each region, whereas an outgoing transition from a composite state represents an outgoing transition from any substate within the composite state. The composite state is exited by 1) reaching the final state in each region, 2) executing a transition from one of the substates within the composite state to an outside state, or 3) executing one of the outgoing transitions from the composite state. Each of the fork and join states is represented by a short, heavy bar. The fork state splits an incoming transition into several unlabeled transitions terminating on states in different regions of a composite state. The join state merges multiple incoming transitions from states of different regions of a composite state into a single unlabeled outgoing transition. The join state cannot be exited unless all of its incoming transitions are executed.

Fig.2 shows an S-SPEC example modeled in a UML state machine. This example demonstrates a simple data transfer application consisting of three entities: a server and two machines available at three SAPs. The service described in the S-SPEC is a server-controlled transfer of data between two machines, in which the

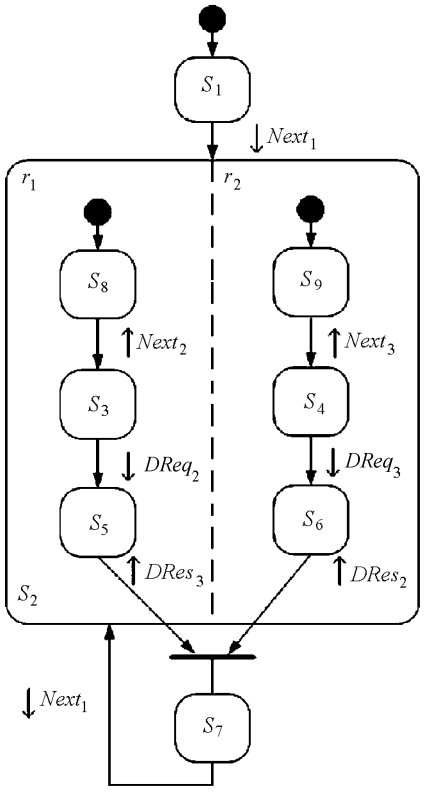

Fig.2. Service specification example modeled in a UML state machine.

users of the two machines exchange their data. The server and the two machines process concurrently. The service is initiated by the server user by issuing a *Next* SP downward command to the sever machine. This service request results in two concurrent upward requests, *Next* SP at SAP_2 and SAP_3, asking the users at the two SAPs to exchange their data. Each of the users at SAP_2 and SAP_3 issues a downward *DReq* SP to request the data from the other user and an upward *DRes* SP as a response for the data requested by the other user. Then, the service cycles back after receiving a *Next* SP downward command from the server user. In this example, if one or both machines have nothing to exchange, then they send a NULL data message. In this example, there are eight simple typical states, three initial states, a join state, and a composite state. The composite state includes two orthogonal regions.

3 Model Definition

UML state machines model concurrency behaviors, and therefore, they can be used without extension to model both service and protocol specifications. In this section, the models used are formally defined in the context of the layered communication system introduced in Section 1. Both service and protocol specifications are modeled using UML protocol state machines.

3.1 Service Specification Model

The service specification described in the UML state machine defines sequences of primitives exchanged between users and processes through the service access points. A service specification includes a concurrent behavior, modeled by composite states that include multiple regions, if two or more service primitives pass through different SAPs simultaneously.

Definition 1. *A service specification S-SPEC is modeled by a UML state machine denoted by a tuple* (S_s, T_s, σ), *where*:

1) S_s *is a non-empty finite set of service states. Each state* $s \in S_s$ *is either a simple or a composite state, as described in Section* 2, *and it is a* 2*-tuple* $\langle id(s), Rg(s)\rangle$, *where* $id(s)$ *is the identifier of state* s *and* $Rg(s)$ *is a finite set of regions in state* s. *Each region* $rg \in Rg(s)$ *is a region in state* s, *such that* $|Rg(s)| = 0$ *for simple states and* $|Rg(s)| > 0$ *for composite states. The region* rg *is a* 2*-tuple* $\langle id(rg), S(rg)\rangle$, *where* $id(rg)$ *is the identifier of region* rg *and* $S(rg)$ *is a finite set of service states included in region* rg. *A simple state can be an initial, final, join, fork, junction, choice, or a typical state as described in Section* 2.

2) T_s *is a finite set of transitions, such that each transition* $t \in Ts$ *is a* 3*-tuple* $\langle tail(t), head(t), SP\rangle$, *where* $tail(t)$ *and* $head(t)$ *are the tail and the head states of* t, *respectively, and* SP *is the service primitive that defines the service event, its type, and the index of the SAP through which the SP passes, which is denoted by* $SAP(SP)$.

3) $\sigma \in S_s$ *is the initial service state.*

For the service specification given in Fig.2, $S_s = \{\langle S_0, \{\}\rangle, \langle S_1, \{\}\rangle, \langle S_2, \{R_1, R_2\}\rangle, \langle S_j, \{\}\rangle, \langle S_7, \{\}\rangle\}$, where $R_1 = \langle r_1, \{S_i, S_3, S_5\}\rangle$, $R_2 = \langle r_2, \{S_k, S_4, S_6\}\rangle$, S_0 is the service specification initial state, S_i is the initial state in region r_1, S_k is the initial state in region r_2, and S_j is the join state that is reached after terminating state S_2. The outgoing transition from state S_1 in the service specification shown in Fig.2 is formally defined as $\langle S_1, S_2, \downarrow Next_1\rangle$.

Definition 2. *The set of SAPs through which the SPs (that can first occur when leaving state* s*) pass is denoted by* $OUT(s)$.

For the example given in Fig.2, $OUT(S_8) = \{2\}$.

Definition 3. *The set of SAPs through which the SPs (that can first occur when reaching composite state* s*) pass is denoted by* $InC(s)$.

For the example given in Fig.2, $InC(S_2) = \{2, 3\}$.

Definition 4. *The set of SAPs through which the SPs included in a region* R *of a composite state pass is denoted by* $OUT(R)$.

For the example given in Fig.2, $OUT(r_1) = \{2, 3\}$.

Definition 5. *A service primitive* P_i *is of type "*$\uparrow$*", denoted as* $\uparrow P_i$, *if the SP is directed upward from the protocol entity PE-SPEC*$_i$ *to SAP*$_i$. *Similarly, a service primitive* P_i *is of type "*$\downarrow$*", denoted as* $\downarrow P_i$, *if the SP is directed downward from the service user at SAP*$_i$ *to the protocol entity PE-SPEC*$_i$.

As shown in Fig.2, the types of the service primitives are denoted by the corresponding upward and downward symbols.

3.2 Protocol Specification Model

The protocol specification describes the specifications of the protocol entities that work together to offer the service that is expressed in the service specification. We assume that each protocol entity is executed by its own processor and that different protocol entities are simultaneously executed by different processors. This constraint implies that none of the composite states in the protocol specifications have multi-regions.

Definition 6. *The protocol entity specification PE-SPEC*$_i$ *is modeled by a UML state machine denoted by a tuple* $(S_{pi}, T_{pi}, \sigma_{pi})$, *where*:

1) S_{pi} *is a non-empty finite set of states of protocol entity* i. *Each state* $s \in S_{pi}$ *is either a simple or a composite state, and it is a* 2*-tuple* $\langle id(s), S(s)\rangle$, *where* $id(s)$ *is the identifier of state* s *and* $S(s)$ *is a finite set of service states included in state* s, *such that* $|S(s)| = 0$

for simple states and $|S(s)| > 0$ for composite states. A simple state can be an initial, final, join, fork, junction, choice, or a typical state as described in Section 2. *It is important to note that the definition of the states in the protocol specification is similar to the corresponding definition for the states in the service specification, which is given in Definition* 1.

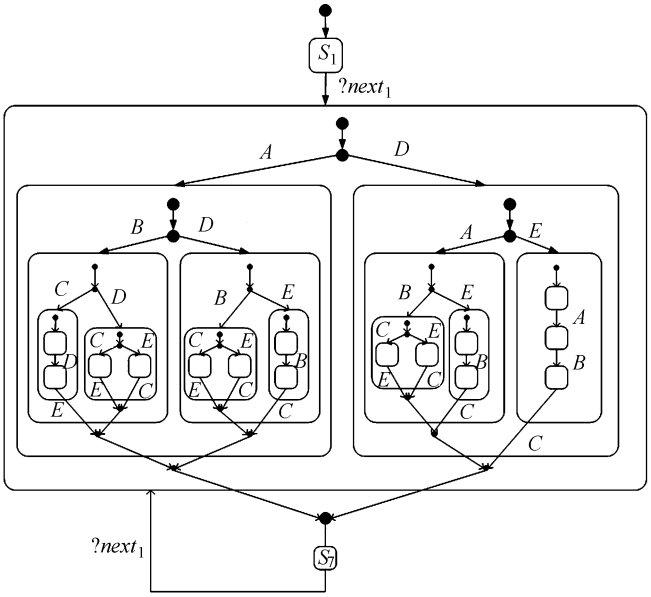

Fig.3. *PE-SPEC*$_2$ corresponding to the S-SPEC given in Fig.2 (*A*: *Next*, *B*: *DReq*/!$dreq_3$, *C*: ?$dres_3$, *D*: ?$dreq_3$, *E*: *DRes*/ !$dres_{1,3}$).

2) *T_{pi} is a finite set of transitions, such that each transition $t \in T_{pi}$ is a* 3*-tuple $\langle tail(t), head(t), E_i \rangle$, where $tail(t)$ and $head(t)$ are the tail and the head states of t, respectively, and E_i is a protocol event that can be either:* 1) *an SP that passes through SAP_i,* 2) *an SP that passes through SAP_i and an event message E sent to PE_j denoted by $!e_j$, or* 3) *an event message E received from PE_j denoted by $?e_j$. The event of the second type is denoted by $E/!e_j$.*

3) *$\sigma_{pi} \in S_{pi}$ is the initial protocol state of the protocol entity specification PE-SPEC$_i$.*

For example, the state S_1 in *PE-SPEC*$_2$, which is given in Fig.3, is formally defined as $\langle S_1, \{\} \rangle$. For the *PE-SPEC*$_2$ given in Fig.3, the outgoing transition from S_1 and the two outgoing transitions from the choice state in the outer composite state are formally defined as follows: $\langle S_1, S_c, ?next_1 \rangle$, $\langle S_k, S_{c1}, Next \rangle$, and $\langle S_k, S_{c2}, ?dreq_3 \rangle$, respectively, where S_c is the outer composite state, S_k is the choice state in the outer composite state, and S_{c1} and S_{c2} are the two inner composite states included in the outer composite states.

Definition 7. *A protocol specification (P-SPEC) describes several simultaneously interacting PE-SPECs, such that there is a one-to-one correspondence that exists between PE-SPECs and SAPs.*

For example, there are three SAPs included in the communication service described by the S-SPEC given in Fig.2, and therefore, three PE-SPECs collaborate to provide the required service. Because of the space limitations, Fig.3 only shows one of these PE-SPECs, which is PE-SPEC$_2$. This paper shows how to synthesize the PE-SPECs from the S-SPEC.

4 UML Protocol State Machine-Based Synthesis Method

To synthesize the specifications of the protocol entities from the service specification, it is required to apply a set of remodeling and optimization rules. Fig.4 provides a synthesis algorithm to obtain the required PE-SPECs in five main steps. The first step is a direct adaptation of the corresponding step in the FSM-based synthesis methods, which were proposed by [11-12, 46], with respect to the context of the synthesis problem that is based on a UML state machine. The 3rd and 5th steps are optimization actions. The key differences between the UML state machine synthesis algorithm and the existing FSM-based synthesis algorithms, which were proposed by [11-12, 46], are in the two main synthesis steps: the second and the fourth. In these two steps, the projected service specification states and transitions are remodeled to form the states and transitions of the protocol specification. The transition synthesis rules proposed here consider the different types of UML state machine states. It is important

PE-SPEC Synthesis Algorithm. Derivation of a protocol specification for distributed concurrent entities from a service specification modeled in a UML protocol state machine.

Input: Service specification modeled using a UML protocol state machine.

Output: PE-SPEC for each of the distributed concurrent entities.

Steps:

1. Project the S-SPEC onto each SAP to obtain the projected protocol specifications (PR-SPECs).
2. Apply transition synthesis rules provided in Table 1 to the transitions of the PR-SPECs to obtain the primary protocol specifications of the entities (PPE-SPECs).
3. Remove ε-transitions and ε-cycles from the PPE-SPECs and apply a state machine reduction technique to obtain the minimized PPE-SPECs.
4. Remodel all composite states with multiple regions using Rules 5 and 6 given in Fig.8 to obtain the protocol specifications of the entities (PE-SPECs).
5. Apply a state machine reduction technique to obtain the minimized PE-SPECs.

Fig.4. Protocol state machine based PE-SPEC synthesis algorithm.

to note that some of these states (e.g., composite, join, fork) are undefined for FSM. The 4th step accounts for remodeling composite states, a problem which is inapplicable for the basic FSM specification model. The five steps are detailed as follows:

Step 1: *Constructing PR-SPECs.* In this step, i PR-SPECs are constructed, where i is the number of distributed concurrent entities, and each PR-SPEC has the same structure as the S-SPEC. Each $PR\text{-}SPEC_i$ has two types of transitions: SP-labeled and unlabeled. The SP-labeled transitions correspond to the S-SPEC transitions assigned to SPs, which are observed at SAP_i. Fig.5 shows the three PR-SPECs that correspond to the S-SPEC given in Fig.2. For example, the outgoing transition from S_1 in $PR\text{-}SPEC_1$ given in Fig.5 is labeled by the SP *Next* because this primitive is observed at SAP_1 as given in the S-SPEC shown in Fig.2. The unlabeled transitions correspond to the S-SPEC transitions assigned to SPs that are either observed at SAP_j, where $i \neq j$, or correspond to S-SPEC unlabeled transitions (e.g., outgoing transitions from initial or join states). For example, the outgoing transition from S_1 in $PR\text{-}SPEC_2$ given in Fig.5 is unlabeled because the corresponding S-SPEC transition is labeled by an SP that is not observed at SAP_2.

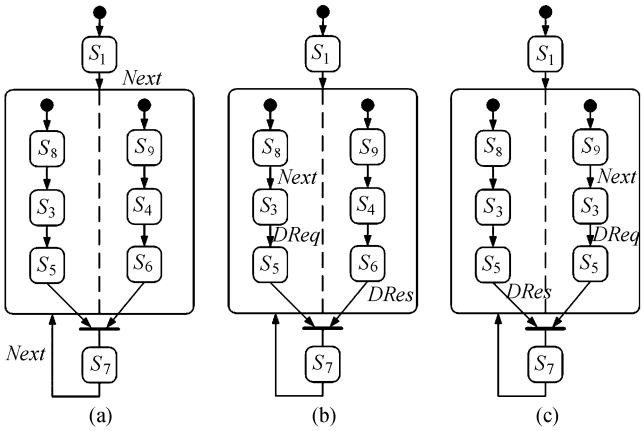

Fig.5. PR-SPECs corresponding to the S-SPEC given in Fig.2. (a) $PR\text{-}SPEC_1$. (b) $PR\text{-}SPEC_2$. (c) $PR\text{-}SPEC_3$.

Step 2: *Applying Transition Synthesis Rules.* Transition synthesis rules are applied to each SP-labeled transition in each PR-SPEC. These rules are summarized in Table 1. For example, Rule 3 is applied to an SP-labeled transition whose destination state is a composite in $PR\text{-}SPEC_i$. This rule states that, according to the formula in the third column of Table 1, we have to find the set of protocol entities x that have to be synchronized with protocol entity i. If the set x is empty, the event E of the transition is unchanged and the corresponding transitions in the other PR-SPECs have to be associated with event ε. Otherwise, the event E of the transition of interest is replaced by $E/!e_x$, which means that a synchronization message has to be sent to each entity in set x. The corresponding transition in each entity in set x has to be associated with a reception event $?e_i$. The corresponding transition in each of the rest of the entities that are unlisted in set x has to be associated with event ε.

Rule 1. In this case, the SP originates from the service user and takes the service back to its initial state. Therefore, a synchronization message must be sent to all of the other protocol entities to synchronize the protocol at the same initial global stable state.

Rule 2. This rule implies that the SP originates from the service user at SAP_i and is taking the service into a state that is outside of the composite state. Therefore, synchronization messages must be sent to all of the protocol entities that issue the SPs that are modeled within the composite state. This scenario would synchronize the protocol and ensure that $PE\text{-}SPEC_i$ does not leave the composite state unless other protocol entities x that are executing concurrently with $PE\text{-}SPEC_i$ leave the same state also. In this case, it is guaranteed that some x protocol entities exist because having multiple regions in a composite state implies the presence of at least another protocol entity that executes concurrently with $PE\text{-}SPEC_i$.

Rule 3. This rule implies that the SP originates from the service user at SAP_i and is taking the service into a composite state. In this case, the SP can be followed

Table 1. Transition Synthesis Rules

Rule ID	Condition (s_1 and s_2 are source and destination states of the transition t of interest)	x	Event E of Transition t in $SPEC_i$	Corresponding Event in $PR\text{-}SPEC_x$	Corresponding Event in Other PR-SPEC$_s$
1	s_2 is a stable initial state of the state machine	all SAPs $- SAP_i$	$E/!e_x$	$?e_i$	Not applicable
2	s_2 is a final state in the composite state cs or s_1 is a substate of cs and s_2 is not a substate of cs or s_1 is a composite state ($s_1 = cs$)	$[\bigcup_{\forall rg \in cs} OUT(rg)] - SAP_i$	$E/!e_x$	$?e_i$	Not applicable
3	s_2 is a composite state	$InC(s_2) - SAP_i$	$E/!e_x$, if $x \neq \varnothing$	$?e_i$	ε
4	s_2 is a simple state	$OUT(s_2) - SAP_i$	E, if $x = \varnothing$	$?e_i$	ε

by any of the SPs that can first occur when reaching the composite state. If all of these SPs are observed at SAP_i, the flow of control is not required to be transferred from *PE-SPEC*$_i$ to another protocol entity or service user, and no actions are needed to be performed at any other PE-SPECs. Otherwise, a synchronization message must transfer the flow of control to the protocol entities that correspond to the SAPs at which the SPs are observed.

Rule 4. This rule implies that the SP originates from the service user at SAP_i and is followed by the occurrence of other SPs that are observed at the same SAP or at other SAPs. If the SP is followed by the occurrence of other SPs that are observed at the same SAP_i, the flow of control is not required to be transferred from *PE-SPEC*$_i$ to another protocol entity or service user, and no actions are needed to be performed at any other PE-SPECs. Otherwise, a synchronization message must transfer the flow of control to the other corresponding protocol entities.

It is important to note that the rules listed in Table 1 only cover the basic scenarios. Any other scenario is a combination of two or more of the basic scenarios. For example, if the source and destination states of a transition are composite, the set x (listed in the third column of Table 1) will be the union of the corresponding sets of rules 2 and 3 (i.e., $x = [\bigcup_{\forall rg \in cs} OUT(rg)] \cup InC(s_2) - SAP_i$). In addition, if the source and destination states of a transition are composite and simple states, respectively, the set x will be the union of the corresponding sets of rules 2 and 4 (i.e., $x = [\bigcup_{\forall rg \in cs} OUT(rg)] \cup OUT(s_2) - SAP_i$).

Fig.6 shows the application of the transition synthesis rules listed in Table 1 on the PR-SPECs given in Fig.5. For example, the event *Next* associated with the outgoing transition from S_1 in *PR-SPEC*$_1$ is replaced with *Next*/!next$_{2,3}$ according to rule 3. In this case, $x = \{2,3\} - \{1\} = \{2,3\}$. The corresponding transitions in *PPE-SPEC*$_2$ and *PPE-SPEC*$_3$ are associated with events ?*next*$_1$. In addition, the event *Data* associated with the outgoing transition from S_6 in *PR-SPEC*$_2$ is replaced with *DRes*/!*dres*$_{1,3}$, according to rules 2 and 4. In this case, the event causes the protocol to leave the composite state and reach state S_7. Therefore, $x = [\bigcup_{\forall rg \in s2} OUT(rg)] \cup OUT(s_7) - SAP_1 = \{2,3\} \cup \{1\} - \{1\} = \{2,3\}$. The corresponding transitions in *PPE-SPEC*$_1$ and *PPE-SPEC*$_3$ are associated with events ?*dres*$_2$.

Step 3: *Removing* ε*-Transitions and* ε*-Cycles.* The resulting PPE-SPECs can include ε-transitions and ε-cycles. These transitions and cycles are no longer needed, and they have to be removed because they do not represent any action required by the protocol entities. In addition, the state machines of the resulting protocol specification can be reduced by applying state machine reduction techniques such as those introduced by [51-52]. Fig.7 shows the resulting minimized PPE-SPECs after applying step 3 to the PPE-SPECs given in Fig.6.

Steps 4 *and* 5: *Remodeling Composite States and Applying Optimization Techniques.* The composite states in the resulting state machines can have multiple regions, which represents concurrency behavior. In the communication system considered, the protocol entities operate concurrently, and this concurrent behavior is modeled in the state machine of the service specification by the multi-region composite state(s). In other words, each protocol entity has its own single processor, which executes concurrently with the processors of the other protocol entities of the communication system. Having multi-region composite state(s) in the specification of the protocol entity implies that the protocol

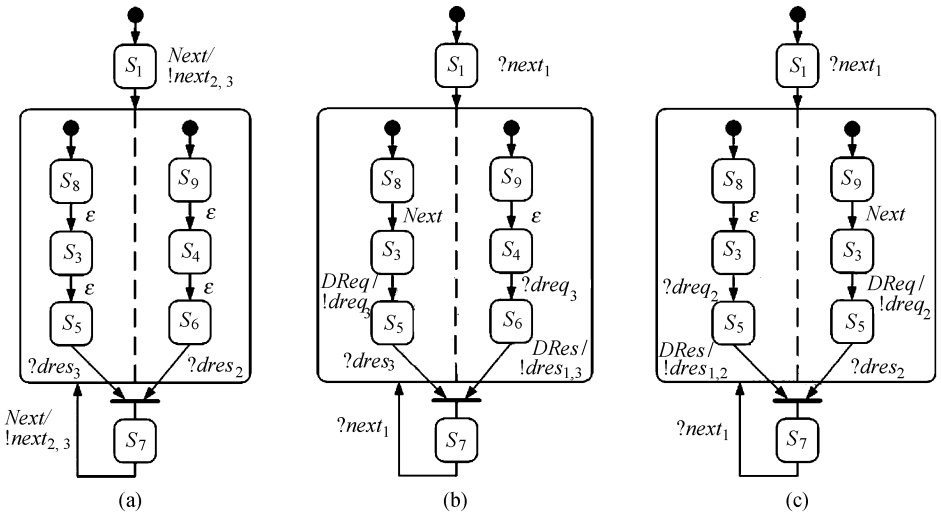

Fig.6. PPE-SPECs corresponding to the PR-SPECs given in Fig.5. (a) *PPE-SPEC*$_1$. (b) *PPE-SPEC*$_2$. (c) *PPE-SPEC*$_3$.

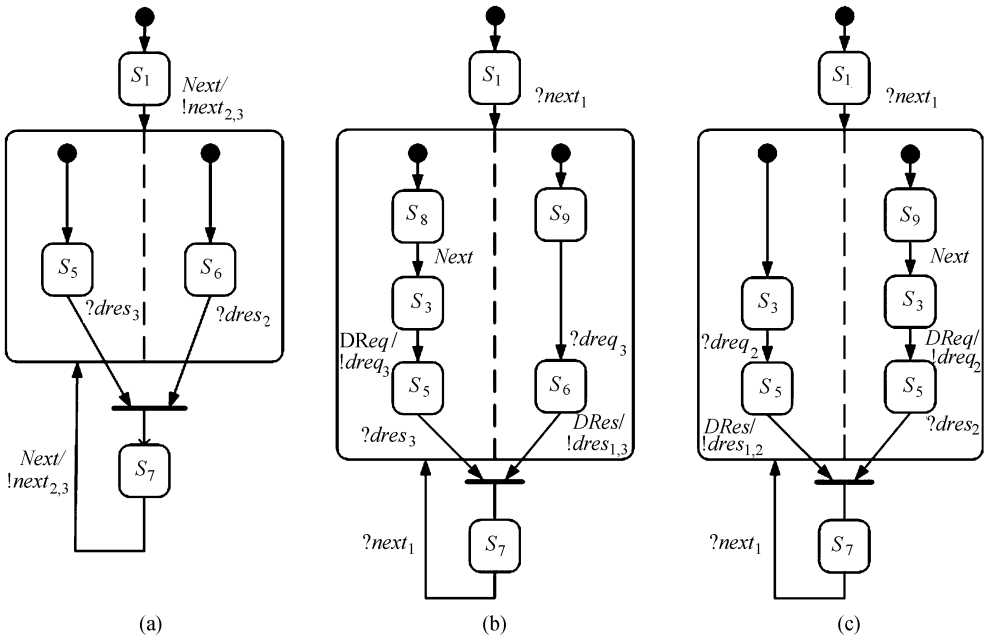

Fig.7. Minimized PPE-SPECs corresponding to the PPE-SPECs given in Fig.6. (a) *PPE-SPEC*$_1$. (b) *PPE-SPEC*$_2$. (c) *PPE-SPEC*$_3$.

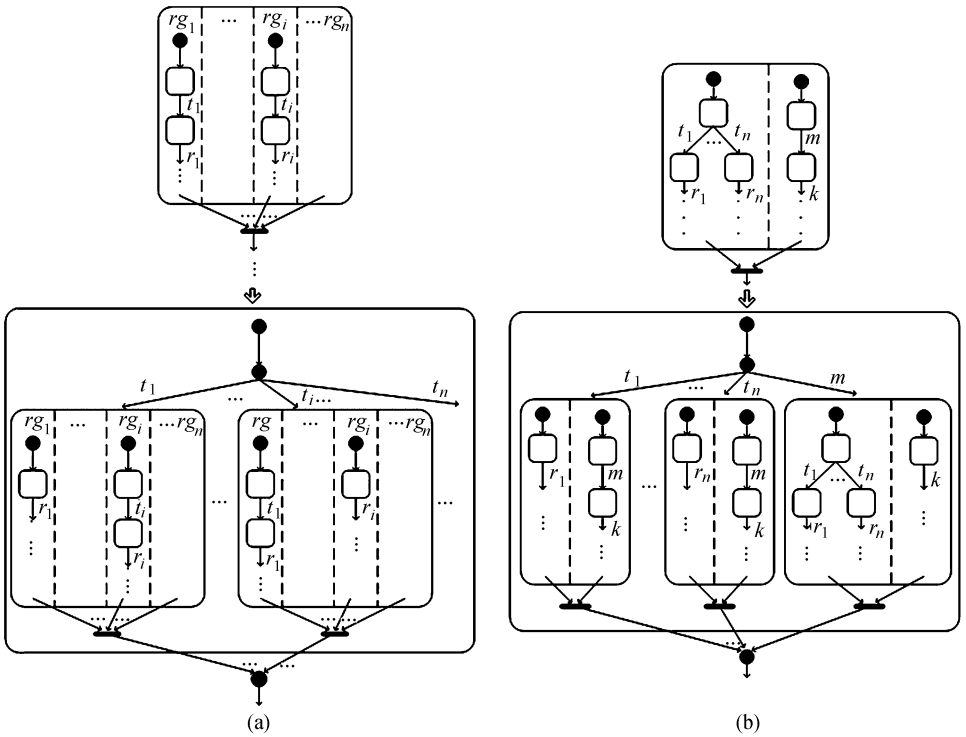

Fig.8. Rules 5 and 6 for remodeling composite states with multiple regions. (a) Rule 5. (b) Rule 6.

entity has multiple processors that operate concurrently, which is not assumed. Therefore, the multi-region composite states of the resulting state machines of the protocol entities have to be remodeled to single-region composite states in such a way that 1) all of the possible orderings of the events are preserved and 2) all of the events are executed.

In any region of a composite state, a stable initial state can have single or multiple outgoing transitions. Rules 5 and 6 (Fig.8) deal with these two cases, respectively. In rule 5, 1) a new choice state is added, 2) the existing n regions are remodeled into n composite

states, 3) each composite state i has the same structure as the original composite state with the exception that the first transition and its destination state in region i are removed, 4) a transition is added from the added choice state to each of the added composite states such that the event associated with a transition that is incoming to composite state i is the same as the event of the transition removed from the composite state i, and 5) a junction state is added to be the destination state of the transitions additionally initiated from the added composite states or their join states. The application of rule 5 results in removing a transition and its destination state from each region. The region is eliminated once it becomes empty or it includes meaningless behavior (i.e., a single unlabeled transition from the initial to the final state). The rule has to be recursively applied to each of the resulting composite states until each composite state has a single region. Rule 6 considers the case when the stable initial state of a region has n multiple outgoing transitions. In this case, the region that contains that initial stable state is remodeled into n composite states, similar to the remodeling performed in rule 5. Moreover, an additional composite state is added to model the case when the first transition in the other region is executed first.

In the case of having nested composite states, rules 5 and 6 are first applied to the most inner composite states. Figs.9∼12 show the resulting *PE-SPEC*$_2$ when the 1st, 2nd, 3rd, and 4th recursive iterations of rule 5 were applied. The resulting *PE-SPEC*$_2$ given in Fig.12 does not include multiple-region composite states. Similar to step 3, a state machine reduction technique can be applied in step 5 to obtain the reduced PE-SPECs. The resulting *PE-SPEC*$_2$ is shown in Fig.3. The rest of the PE-SPECs can be obtained similarly but are not shown here due to the space limitation.

5 Space and Time Complexity Analysis

The space and time complexity of the proposed synthesis algorithm is determined by finding the number of states and transitions created by the algorithm and analyzing the time and space required to create and store the protocol specifications.

5.1 Space Complexity Analysis

Analysis of the space complexity requires finding the number of states and transitions in the resulting protocol specification. The number of states and transitions in each PR-SPEC created in step 1 of the algorithm given in Fig.4 is equal to the number of states and transitions in the S-SPEC. Given that s and t are the number of states and transitions, respectively, in

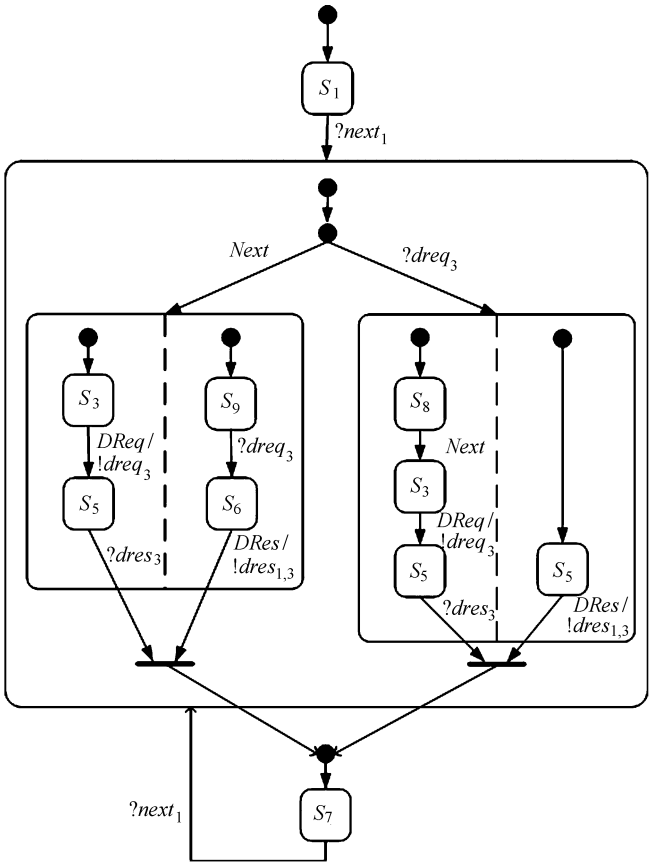

Fig.9. Application of rule 5 (first iteration) to obtain *PE-SPEC*$_2$.

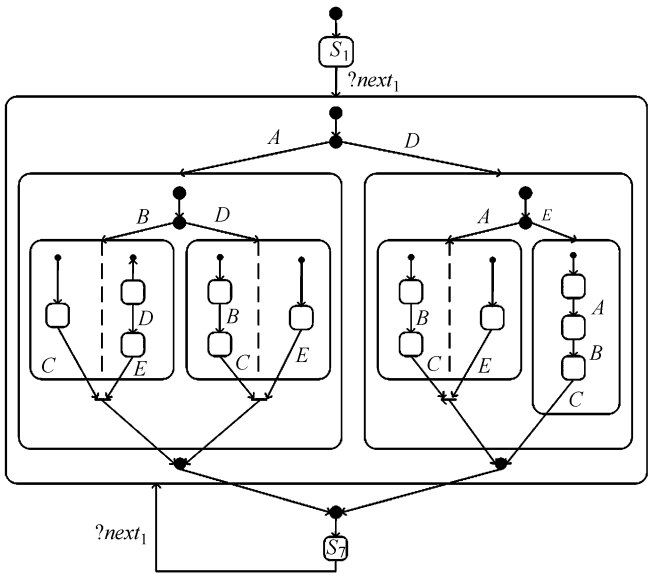

Fig.10. Application of rule 5 (second iteration) to obtain *PE-SPEC*$_2$ (*A*: *Next*, *B*: *DReq*/!$dreq_3$, *C*: ?$dres_3$, *D*: ?$dreq_3$, *E*: *DRes*/!$dres_{1,3}$).

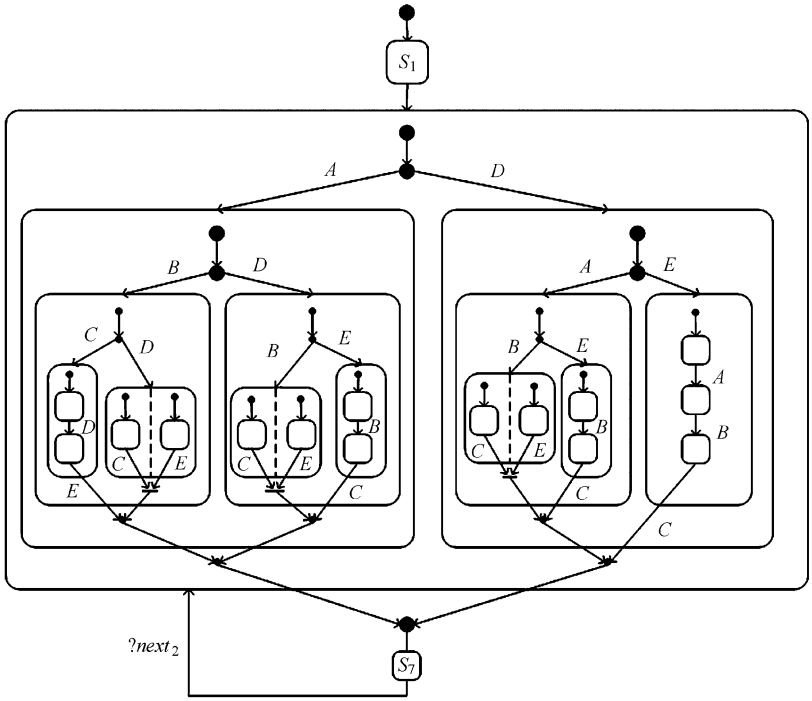

Fig.11. Application of rule 5 (third iteration) to obtain *PE-SPEC*$_2$.

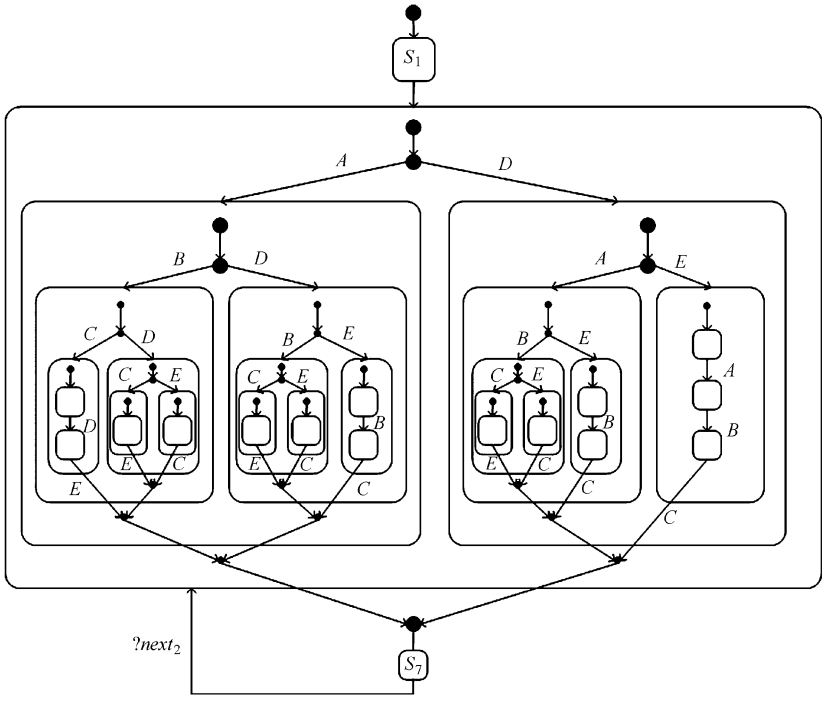

Fig.12. Application of rule 5 (fourth iteration) to obtain *PE-SPEC*$_2$.

S-SPEC, and that n is the number of SAPs provided in the system, the number of states and transitions created in step 1 of the algorithm is $n \times s$ and $n \times t$, respectively. Steps 2 and 3 of the algorithm do not add any states and transitions to the created models. Instead, step 2 requires parsing each transition to determine its event, and step 3 requires removing the ε-transitions and ε-cycles. The number of states and transitions removed in step 3 depends on the need for the synchronization messages introduced in step 2 and, in some cases, none

of such ε-transitions and ε-cycles exists. Therefore, the number of states and transitions existing after executing step 3 of the algorithm is still bounded by $n \times s$ and $n \times t$, respectively.

In step 4, rules 5 and 6 are applied recursively until each composite state has a single region. In this case, the resulting structure has paths from the initial state to the final state of the composite state such that each path includes a possible trace of *all transitions* in the paths that are corresponding to the S-SPEC parallel paths (i.e., paths that include transitions in different regions). Formally, given a composite state k of m_k regions, such that each region has a single path from the region's initial state to its final state, when rule 5 is applied recursively, the number of transitions in each of the resulting paths equals $\sum_{i=1}^{m_k} n_{i,k}$, where $n_{i,k}$ is the number of transitions in the i-th region of the k-th composite state.

Rule 5 remodels m paths that exist in m regions of composite state k into corresponding paths $P_{k,m}$, such that these paths cover all possible traces of the transitions in the m paths. The value of $P_{k,m}$ is equal to the number of different combinations of transitions in the m paths, such that the order of the transitions in each of the m path is preserved. In the case of having a composite state k of only two regions, where one of the regions has $n_{1,k}$ transitions and the other one has $n_{2,k}$ transitions, the number of possible combinations is calculated by using the following formula[53]:

$$\begin{aligned} P_{k,2} =& C(n_{1,k} + n_{2,k}, n_{2,k}) \\ =& \binom{n_{1,k} + n_{2,k}}{n_{2,k}} = \frac{(n_{1,k} + n_{2,k})!}{(n_{1,k}!)(n_{2,k}!)}. \end{aligned} \tag{1}$$

For example, there are two regions in the composite state included in *PPE-SPEC*$_2$, which is shown in Fig.7. One of the regions has three transitions with events, and the other has two transitions with events. Therefore, the number of paths obtained by applying rule 5 equals $C(2+3, 2) = 10$, as shown in Fig.12.

Generally, given a composite state k with m regions, where each region has a single path, the number of paths $P_{k,m}$, produced using rule 5 is formally calculated as follows:

$$P_{k,m} = \frac{\Big(\sum_{i=1}^{m_k} n_{i,k}\Big)!}{\prod_{i=1}^{m_k}(n_{i,k}!)}. \tag{2}$$

Proof. To prove this formula using mathematical induction, both basic and inductive steps have to be proven as follows.

Basic Step. The minimum number of parallel paths is two. In this case, using (2),

$$P_{k,2} = \frac{\Big(\sum_{i=1}^{2} n_{i,k}\Big)!}{\prod_{i=1}^{2}(n_{i,k}!)} = \frac{(n_{1,k} + n_{2,k})!}{(n_{1,k}!)(n_{2,k}!)},$$

which is equal to the result obtained using (1) and proven by [53].

Inductive Step. In the inductive step, assuming that $P_{k,m}$ is correct, we have to prove that $P_{k,m+1}$ is also correct. As discussed earlier in this section, given m_k regions, each of the $P_{k,m}$ paths produced using rule 5 consists of all transitions in all m_k regions, which equals $\sum_{i=1}^{m_k} n_{i,k}$. When adding region m+1, the transitions in this additional region can be interleaved with the transitions in any of the $P_{k,m}$ paths. Using (1), when the transitions of the path in region $m+1$ are interleaved with the transitions of any region j, the number of resulting paths equals $C\left(\sum_{i=1}^{m_k} n_{i,k} + n_{m+1,k}, n_{m+1,k}\right)$. As a result, considering all $P_{k,m}$ paths, the total number of produced paths is calculated as follows:

$$\begin{aligned} P_{k,m+1} =& P_{m,k} \times C\Big(\sum_{i=1}^{m_k} n_{i,k} + n_{m+1,k}, n_{m+1,k}\Big) \\ =& \frac{\Big(\sum_{i=1}^{m_k} n_{i,k}\Big)!}{\prod_{i=1}^{m_k}(n_{i,k}!)} \times \frac{\Big(\sum_{i=1}^{m_k} n_{i,k} + n_{m+1,k}\Big)!}{\Big(\sum_{i=1}^{m_k} n_{i,k}\Big)! \times n_{m+1,k}!} \\ =& \frac{\Big(\sum_{i=1}^{m_k} n_{i,k}\Big)!}{\prod_{i=1}^{m_k}(n_{i,k}!)} \times \frac{\Big(\sum_{i=1}^{m_k+1} n_{i,k}\Big)!}{\Big(\sum_{i=1}^{m_k} n_{i,k}\Big)! \times n_{m+1,k}!} \\ =& \frac{\Big(\sum_{i=1}^{m_k+1} n_{i,k}\Big)!}{\prod_{i=1}^{m_k}(n_{i,k}!) \times n_{m+1,k}!} = \frac{\Big(\sum_{i=1}^{m_k+1} n_{i,k}\Big)!}{\prod_{i=1}^{m_k+1}(n_{i,k}!)}, \end{aligned}$$

which is equal to $P_{k,m+1}$ found by (2). □

The space complexity of applying rule 5 to a composite state k is bounded by the number of produced transitions and states. The number of produced states and transitions is itself bounded by the multiplication result of the number of resulting paths and the number of transitions in each path (i.e., $P_{k,m} \times \sum_{i=1}^{m_k} n_{i,k}$). Rule 5 produces states and transitions such that there is a direct correspondence between the number of states and transitions. That is, except for the produced junction states, each resulting state has a unique incoming transition. Therefore, the number of states produced

by rule 5 is also bounded by $P_{k,m} \times \sum_{i=1}^{m_k} n_{i,k}$. It is important to note that rule 6 produces paths that satisfy a more constrained condition than that which is satisfied by the paths produced by rule 5, because rule 6 deals with some paths (i.e., paths that start with a choice state) that only one of them will be executed. These are the paths that exist within the same region. The transitions of these paths will not be interleaved with each other. Instead, they will only be interleaved with the paths in the other region(s). As a result, the space required by the application of step 4 of the synthesis algorithm is bounded by the space required by the application of rule 5.

Finally, step 5 of the proposed synthesis algorithm does not introduce any further states and transitions and, in some cases, it does not lead to any reduction in the number of states and transitions. As a result, given an S-SPEC with sp SAPs, t_1 transitions that are not included in composite states, and cs composite states, the space complexity of the algorithm, in terms of the number of transitions, is bounded by $sp \times (t_1 + t_2)$, where $t_2 = \sum_{k=1}^{cs} \left(P_{k,m} \times \sum_{i=1}^{m_k} n_{i,k} \right)$. In addition, based on the earlier discussion regarding the correspondence between the number of states and transitions produced by rule 5, the space complexity of the algorithm, in terms of the number of states, is bounded by $sp \times (s_1 + t_2)$, where s_1 is the number of S-SPEC states that are not included in composite states. As a result, the total space complexity, in terms of both numbers of states and transitions, is bounded by $sp \times (s_1 + t_1 + t_2)$.

5.2 Time Complexity Analysis

The time complexity of the proposed synthesis algorithm is determined by the number of states and transitions that are produced in each step and every iteration of the application of rules 5 and 6. The number of states and transitions produced in each step is already discussed in Subsection 5.1. However, the number of states and transitions produced in each iteration of the application of rules 5 and 6, which has not yet been discussed, is described as follows.

In step 4, the composite states are remodeled using rules 5 and 6. Given a composite state cs of r_{cs} regions, in iteration i of the application of rule 5, $C_{cs}(i) = r_{cs}$ composite states are created, as shown in Fig.8. Except for a removed transition, each of the new r_{cs} composite states has the same structure of the original composite state that existed in iteration $i-1$, which means that, in iteration i, the number of created states and transitions from a composite state cs, which existed in iteration $i-1$, is bounded by $C_{cs}(i-1) \times sc_{cs}$ and $C_{cs}(i-1) \times tc_{cs}$, respectively, where sc_{cs} and tc_{cs} represent the number of states and transitions included in the composite state cs respectively. Therefore, given cs composite states in iteration $i-1$, the number of states produced in iteration i is bounded by $S(i) = \sum_{k=1}^{cs} C_k(i-1) \times sc_k$. Rule 5 is applied until each composite state has a single region. In each iteration, a transition is removed from one of the regions until the region becomes empty or it includes meaningless behavior (i.e., a single unlabeled transition from the initial to the final state). This terminating condition requires the number of applied iterations to be bounded by the total number of transitions included in the original composite state t that exists in PPE-SPEC. Consequently, the total number of states produced in all iterations is bounded by $\sum_{i=1}^{t} S(i)$. As discussed in Subsection 5.1, the number of transitions produced by rule 5 is bounded by the same equation that is used for the number of states. In addition, the discussion in Subsection 5.1 shows that the process of applying rule 5 is more complex than that which pertains to rule 6.

As a result, given an S-SPEC with sp SAPs, t_1 transitions that are not included in composite states, cs composite states, and t transitions that are included in composite states, the time complexity of the first four steps of the algorithm, in terms of the number of transitions, is bounded by $sp \times (t_1 + t_2)$, where $t_2 = \sum_{i=1}^{t} S(i)$, and the time complexity in terms of number of states, is bounded by $sp \times (s_1 + t_2)$, where s_1 is the number of S-SPEC states that are not included in composite states. The total time complexity, in terms of both numbers of states and transitions, is bounded by $sp \times (s_1 + t_1 + t_2)$. The time complexity of step 5 of the algorithm depends on the time complexity of the selected reduction technique.

6 Proofs of Correctness

Proving the correctness of the synthesis method is needed to support the claim that the resulting protocol specification does not require any further validation. Both semantic and syntactic correctness have to be proven.

6.1 Semantic Correctness

Proving semantic correctness requires proving that the resulting protocol specification provides the same service specified in the S-SPEC with the same possible orderings of the SPs.

Definition 8. *$\mu(SM)$ is the collection of all of the possible traces of SPs modeled in the state machine SM.*

For example, ($?next_1$, A, B, C, D, E, $?next_1$) is a possible trace of SPs modeled in the *PE-SPEC*$_2$ given in Fig.3.

Lemma 1. $\mu(PPE\text{-}SPEC_i) = \mu(PR\text{-}SPEC_i)$.

Proof. By definition, *PPE-SPEC*$_i$ has the same

structure of $PR\text{-}SPEC_i$, and step 2 of the synthesis algorithm does not add, remove, or change any SP associated with any transition in PR-SPEC; it only adds synchronization messages. Therefore, the possible traces of SPs in $PPE\text{-}SPEC_i$ are identical to the corresponding traces in $PR\text{-}SPEC_i$. □

Definition 9. $\prod_i(S\text{-}SPEC)$ *is the collection of all of the possible traces of SPs observed at* SAP_i *and specified in S-SPEC.*

For example, (*Next*, *DReq*) is a possible trace of SPs observed at SAP_2 for the S-SPEC given in Fig.2.

Lemma 2. $\prod_i(S\text{-}SPEC) = \mu(PPE\text{-}SPEC_i)$.

Proof. Both S-SPEC and PR-SPEC$_i$ have the same structures, and by the definition, $PR\text{-}SPEC_i$ is the projection of S-SPEC at SAP_i, which implies that $\prod_i$(S-SPEC) $= \mu(PR\text{-}SPEC_i)$. Therefore, according to Lemma 1, $\prod_i$(S-SPEC) $= \mu(PPE - SPEC_i)$. □

Definition 10. $\sum_{\forall i \in \text{SAPs}} \prod_i(S\text{-}SPEC)$ *is the collection of all of the possible traces of SPs observed at each SAP in the communication system.*

For example, this collection includes all possible traces of SPs observed at each of SAP_1, SAP_2, and SAP_3, for the S-SPEC given in Fig.2.

Definition 11. $\sum_{\forall i \in SAPs} \mu(PPE\text{-}SPEC_i)$ *is the collection of all of the possible traces of SPs in each* $PPE\text{-}SPEC_i$.

For example, this collection includes all possible traces of SPs in each of $PPE\text{-}SPEC_1$, $PPE\text{-}SPEC_2$, and $PPE\text{-}SPEC_3$, which are shown in Fig.7.

Lemma 3. $\sum_{\forall i \in SAPs} \prod_i(S\text{-}SPEC) = \sum_{\forall i \in SAPs} \mu(PPE\text{-}SPEC_i)$. *That is, all of the SPs specified in S-SPEC are also specified in PPE-SPECs and vice versa.*

Proof. The above equality is a direct result of the relation between Definitions 10 and 11 and the equality given in Lemma 2. □

Definition 12. $\bigcup(PPE\text{-}SPEC_i)$ *is a state machine derived from* $PPE\text{-}SPEC_i$ *by replacing each reception of a synchronization message by its corresponding SP.*

For example, $\bigcup(PPE\text{-}SPEC_1)$ is derived from $PPE\text{-}SPEC_1$, which is given in Fig.7, by replacing the $?dres_3$ message by the *DRes SP* and replacing the $?dres_2$ message by the *DRes* SP.

Lemma 4. $\mu(\bigcup(PPE\text{-}SPEC_i))$ *includes all of the legal interleavings of SPs at* SAP_i *with respect to some SPs observed at other SAPs.*

Proof. The reception of a synchronization message in $PPE\text{-}SPEC_i$ implies that a corresponding SP is observed at another SAP_j, where $i \neq j$. Therefore, having an SP A preceeding $?b$ in $PPE\text{-}SPEC_i$ implies that A precedes B in both $\mu(\bigcup(PPE\text{-}SPEC_i))$ and $\mu(S\text{-SPEC})$. As a result, the order of SP occurrences in $\mu(\bigcup(PPE\text{-}SPEC_i))$ is consistent with the order of SP occurrences in $\mu(S\text{-SPEC})$. □

Lemma 5. *All of the traces in* $\mu(S\text{-}SPEC)$ *are preserved in* $\sum_{\forall i \in SAPs} \mu(\bigcup(PPE\text{-}SPEC_i))$.

Proof. Any pair of adjacent SPs A_iB_j in $\mu(S\text{-SPEC})$ is observed either at the same SAP (i.e., $i = j$) or at different SAPs (i.e., $i \neq j$). According to the transition synthesis rules given in Table 1, when $i = j$, both SPs appear adjacent in $\mu(\bigcup(PPE\text{-}SPEC_i))$. When $i \neq j$, according to the transition synthesis rules, the pair of SPs is modeled by contiguous events $A?b_j$, which is represented in $\mu(\bigcup(PPE\text{-}SPEC_i))$ by AB. Therefore, in both cases, the same pair of SPs appears in $\sum_{\forall i \in \text{SAPs}} \mu(\bigcup(PPE\text{-}SPEC_i))$ at least once. □

Lemma 6. *Assume that the composite states before and after applying rule* 5 *or rule* 6 *are denoted by* CS_0 *and* CS_1*, respectively. Given the fact that the events modeled in the composite state are executed by a protocol entity of a single processor,* CS_0 *and* CS_1 *have an identical behavior* (*i.e., they provide the same service with the same event ordering*).

Proof. When rule 5 is applied, CS_1 initially allows for the execution of any of the first executable transitions in the regions of CS_0. When any of these transitions t is executed, the transition t in CS_1 will be followed by the transition that follows t in the same region or by any of the other first executable transitions in the other regions of CS_0. Except for this modification, which does not alter the orderings of the transition execution, the ordering of the transitions in the regions of CS_0 and CS_1 are identical. This structure means that both CS_0 and CS_1 have the same possible event ordering, where the events are associated with transitions. CS_1 has an added choice state that results in executing only one of the added composite states. However, each of these composite states is identical to the original composite state CS_0 with the exception of the removed transition from one of the regions. This transition is added and is to be executed before any of the transitions in the inner composite state. This addition guarantees that, when applying rule 5, CS_1 has the same behavior as CS_0. The same argument applies for rule 6. In this case, the first executable transitions in the regions of CS_0 are the outgoing transitions from the choice state and the first executable transitions in each of the other regions. When rule 6 is applied, each of the first executable transitions t is followed by the transition that follows t in the same region or by any of the other first executable transitions in the other regions of CS_0. □

Theorem 1. *The protocol entities derived using the synthesis algorithm introduced in this paper are semantically correct.*

Proof. Lemmas 3 and 5 imply that the PPE-SPECs

resulting from applying step 2 of the synthesis algorithm provide the same service modeled in the S-SPEC with the same possible orderings of the SPs. This implies that the PPE-SPECs are semantically correct. The correctness of the protocol entities derived in steps 3 and 5 is the result of the correctness of the used reduction, and ε-transitions and ε-cycles removal techniques. Finally, Lemma 6 implies that the protocol entities derived in step 4 are semantically correct with respect to those obtained in step 3. As a result, the protocol entities resulting from the application of the synthesis algorithm proposed in this paper are semantically correct. □

6.2 Syntactic Correctness

Proving the syntactic correctness requires proving that the resulting protocol specification is free of syntactic design errors, including unspecified receptions, deadlocks, and livelocks.

Lemma 7. *The synthesized protocol specification is deadlock-free.*

Proof. Deadlock errors occur when the protocol is at a non-final state, all of the channels are empty, and no transmission transition is specified. In other words, a deadlock occurs when the protocol is at a state in which all of its outgoing transitions are associated with receiving events for messages that are not to be sent by any other protocol entity. This case cannot happen in the synthesized protocol because the transition synthesis rules are based on a cause and effect principle. In other words, the occurrence of a SP at SAP_i will cause either the sending of synchronization messages to one or more of the other protocol entities or the prevention of any protocol message from being sent. The first case causes the other corresponding protocol entities to receive the synchronization message and send the following SP accordingly. The second case occurs only when reaching a final state or when the next occurring SP is observed at the same SAP. As a result, when the protocol is not at a final state, the transition synthesis rules cause the protocol entities to always either issue SPs or exchange synchronization messages until all of the protocol entities reach their final states. Therefore, the protocol entities resulting from applying the transition synthesis rules are free of deadlock errors. In addition, rules 5 and 6 do not cancel any of the synchronization messages. Instead, they flatten the composite states to remove the multiple regions without affecting the services and their order (Lemma 6). Therefore, the resulting protocol entities after applying step 4 of the synthesis algorithm are also free of deadlock errors. The correctness of the protocol entities obtained in steps 3 and 5 depends on the correctness of the applied techniques. □

Lemma 8. *The synthesized protocol specification is livelock-free.*

Proof. A livelock error occurs when the protocol entities exchange messages that are meaningless with respect to the provision of the desired service. The resulting protocol entities include two types of events: SPs and synchronization messages. The SP events are required to provide the service specified in the S-SPEC, and the synchronization messages are required to synchronize the SPs between the distributed protocol entities and enforce the order of the SPs as described in the S-SPEC. As a result, all of the messages exchanged between the protocol entities are meaningful for the provision of the desired service, and therefore, the protocol entities are free of livelock errors. □

Lemma 9. *The synthesized protocol specification is free of unspecified reception errors.*

Proof. An unspecified reception error occurs when a protocol entity sends a synchronization message to another protocol entity that cannot reach a state at which the reception of this message is specified. The transition synthesis rules guarantee that whenever a synchronization message is sent from a protocol entity i at a global state z to another protocol entity j, the protocol entity j will not leave the state corresponding to z unless it receives the sent message. This construct occurs because whenever the transition synthesis rules specify a message to be sent, they specify the corresponding message to be received by all of the protocol entities at which the message is expected to be received. In the case of concurrency (modeled in the composite states that have multiple regions in the S-SPEC), rule 5 and rule 6 remodel each composite state of multiple regions in PPE-SPEC with nested composite states with single regions so that each possible interleaving of the events in different regions is represented in the resulting nested composite states. This restructuring implies that any path from the initial state to the final state of the composite state includes all of the possible receptions modeled previously in different regions. Lemma 6 guarantees the correct ordering of these receptions with respect to their preceding SP labeled transitions, which guarantees that the remodeled composite states will not cause any unspecified reception errors. As a result, the resulting synthesized protocol specifications are free from unspecified reception errors. □

7 Application

In this section, we provide a real application for the proposed synthesis method and discuss the practical limitations and lessons learned by applying the method.

7.1 VoIP Call Establishment Application

We demonstrate the application of the synthesis method to the specification of the call establishment service of the H.323 standard① used for the transmission of real-time audio, video, and data communications over packet-based networks. The version of the service adopted by VoIP② for call signaling includes a gatekeeper G and two endpoints, denoted as $P1$ and $P2$, all processing simultaneously. The gatekeeper is a central point for all of the calls within its zone and provides call control services for registered H.323 endpoints. The service starts when $P1$ sends an admission request ARQ to G and G responds by sending back an admission confirmation ACF. Then, $P1$ sends a setup message to G, which in turn forwards the message to $P2$. $P2$ then sends a call processing message CP to G. In response, G forwards the call processing message CP to $P1$, and concurrently, $P2$ sends an admission request message ARQ to G. G replies to $P2$ by sending an admission confirmation message ACF. $P2$ responds with an alerting message to G, which forwards the message to $P1$. After that, G sends the alerting message to $P1$, and concurrently, the handset at $P2$ can be picked, which results in a connect message CON being sent from $P2$ to G. This connect message is forwarded from G to $P1$. We transformed the service written in English into a UML protocol state machine. Fig.13 shows the service specification modeled in the UML protocol state machine and the corresponding protocol specification derived from S-SPEC using the proposed synthesis method.

7.2 Limitations and Lessons

One of the key limitations of the proposed synthesis

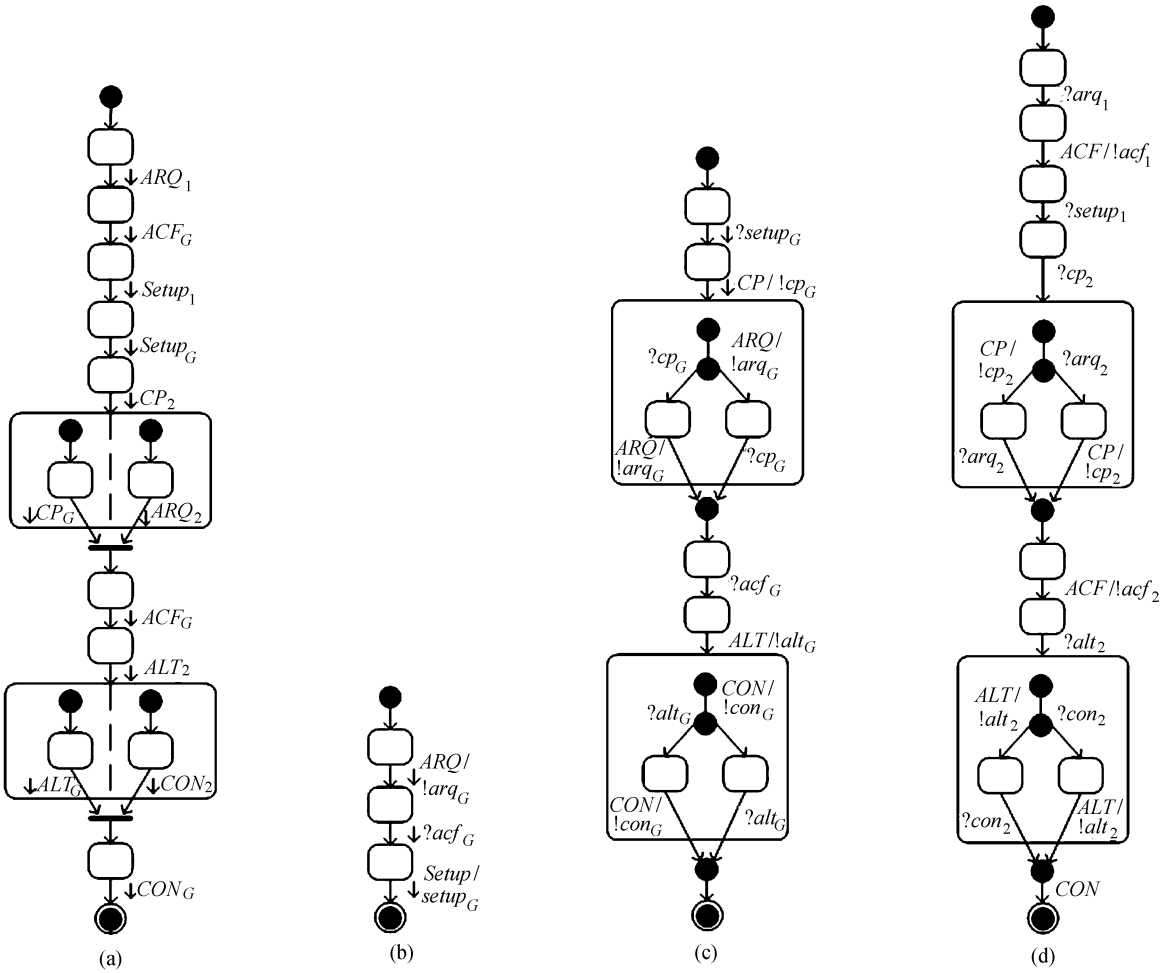

Fig.13. S-SPEC and the synthesized PE-SPECs of the call establishment service of the H.323 standard. (a) S-SPEC. (b) $PE\text{-}SPEC_{P1}$. (c) $PE\text{-}SPEC_{P2}$. (d) $PE\text{-}SPEC_G$.

①Telecommunication standardization sector. http://www.itu.int/ITU-T/index.html, July 2011.

②VoIP protocols: H.323 call flow. http://toncar.cz/Tutorials/VoIP/VoIP_Protocols_H323_Call_Flow.html, June 2011.

method is that, as it currently is, it is inapplicable to real applications that have timing constraints or security conditions. To consider such widely used applications, the S-SPEC model must be extended to specify the timing and security constraints and the synthesis rules have to be modified to derive P-SPEC that complies with the specified constraints. Another practical limitation for the proposed method is that, as discussed in Section 5, the application of rules 5 and 6 causes the P-SPEC to include large number of states and transitions. This problem is partially solved by the application of step 5 of the algorithm, given in Fig.4, in which it is suggested to apply an existing state reduction technique. In addition, the application of rules 5 and 6 is systematic and can be automated, which eliminates the chances of making mistakes in dealing with the growing number of states and transitions in each recursive iteration of the application of rules 5 and 6. Finally, we noticed that most of the real applications in which the service specification is modeled using the UML state machine do not include many states and transitions inside the composite states. For example, the S-SPEC given in Fig.13 includes a single transition with SP in each region of a composite state. We believe that the reason for having few states and transitions inside the composite states of the S-SPECs of most of the real applications that we went through is that, typically, the designers of the communication systems tend to reduce the complexity of the service specifications by minimizing the number of concurrent behaviors. As a result, although the proposed algorithm produces a huge number of states and transitions, in some cases, we found that these cases are rare in real applications.

8 Conclusions and Future Work

This paper proposes a method to synthesize the communication protocol specification of concurrent protocol entities from a service specification modeled in a UML protocol state machine. The synthesis method solves a problem faced by object-oriented communication system developers who start their work by designing a protocol specification. The protocol specification is typically complex and difficult to verify against the service specification, especially when the protocol specification includes concurrent behaviors. On the other hand, in contrast to a protocol specification, a service specification is written at a highly abstract level, which cannot be converted to object-oriented code using the existing code generation tools. Using the proposed synthesis method, the developers can start their design process by formalizing the service specification into a UML protocol state machine and using the synthesis method to automatically derive the protocol specification. The existing tools can then be applied to generate the required code. This paper proves that the derived protocol specification is syntactically and semantically correct, and therefore, it does not require any further verification. To demonstrate the usefulness of the synthesis method, we applied it to synthesize the protocol specification of a real adopted application used in Internet calling. The synthesis method can be extended by considering several factors such as the timing constraints, the security of the exchanged messages, and the reliability of the transmission medium.

References

[1] Lai R, Jirachiefpattana A. Communication Protocol Specification and Verification. Springer-Verlag, 1998.

[2] Probert R, Saleh K. Synthesis of communication protocols: Survey and assessment. *IEEE Transactions on Computers*, 1991, 40(4): 468-476.

[3] Khoumsi A, Bochmann G V, Dssouli R. On specifying services and synthezing protocols for real-time applications. In *Proc. the 14th of IFIP International Symposium on Protocol Specification, Testing and Verification*, June 1994, pp.185-200.

[4] Kakuda Y, Nakamura M, Kikuno T. Automated synthesis of protocol specifications from service specifications with parallely executable multiple primitives. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 1994, E77-A(10): 1634-1645.

[5] Bista B, Shiratori N. Construction of a multiple entities communication protocol by compositional approach. In *Proc. the 12th Int. Workshop on Database and Expert Systems Applications*, September 2001, pp.162-166.

[6] Maneerat N, Varakulsiripunth R, Seki D, Yoshida K, Takahashi K, Kato Y, Bista B B, Shiratori N. Composition method of communication system specifications in asynchronous model and its support system. In *Proc. the 9th IEEE International Conference on Networks*, October 2001, pp.64-69.

[7] Yamaguchi H, El-Fakih K, von Bochmann G, Higashino T. Protocol synthesis and re-synthesis with optimal allocation of resources based on extended Petri nets. *Distributed Computing*, 2003, 16(1): 21-35.

[8] Maneerat N, Varakulsiripunth R, Bista B, Takahashi K, Kato Y, Shiratori N. Composition of service and protocol specifications in asynchronous communication system. *IEICE Transactions on Information and Systems*, 2004, E87-D(10): 2306-2317.

[9] Al Dallal J. Automatic synthesis of timed protocol specifications from service specifications. *WSEAS Transactions on Computers*, 2006, 5(1): 105-112.

[10] Stakhanova N, Basu S, Zhang W, Wang X, Wong J. Specification synthesis for monitoring and analysis of MANET protocols. In *Proc. the 21st AINA Workshops*, May 2007, Vol.1, pp.183-187.

[11] Al Dallal J, Saleh K. Service-oriented synthesis of distributed and concurrent protocol specifications. *Journal of Computer Systems, Networks, and Communications*, 2008, Article No.794960.

[12] Al Dallal J, Saleh K. State-expansion-based techniques for synthesizing concurrent protocol specifications in distributed systems. *International Journal of Communication Systems*, 2012, in press.

[13] Fayed M, Laitinen M. Transition to Object-Oriented Software Development (1st edition), Wiley, 1998.

[14] Jepsen T, Anjum F, Bhat R, Jain R, Sharma A, Tait D. Java in Telecommunications: Solutions for Next Generation Networks. Wiley, 2001.

[15] Schmidt D, Huston S. C++ Network Programming: Systematic Reuse with ACE and Frameworks (Vol.2). Addison-Wesley Professional, 2002.

[16] Dezani-ciancaglini M, Mostrous D, Yoshida N, Drossopoulou S. Session types for object oriented languages. In *Proc. the 6th European Conference on Object-Oriented Programming*, July 2006, pp.328-352.

[17] Herzberg D, Reichert T. Software engineering for telecommunications systems. In *Wiley Encyclopedia of Computer Science and Engineering*, Wah B W (ed.), Wiley, 2009.

[18] Hu R, Yoshida N, Honda K. Session-based distributed programming in Java. In *Proc. the 22nd European Conference on Object-Oriented Programming*, July 2008, pp.516-541.

[19] Noor A. Distributed Java mobile information system. *Communications of the IBIMA*, 2009, 10: 127-132.

[20] Porres I, Rauf I. From nondeterministic UML protocol statemachines to class contracts. In *Proc. the 3rd ICST*, April 2010, pp.107-116.

[21] Object management group. Documents associated with UML Version 2.3, http://www.omg.org/spec/UML/2.3/, July 2011.

[22] Björklund D, Lilius J, Porres I. Towards efficient code synthesis from statecharts. In *Proc. Workshop of the pUML-Group*, Oct. 2001, pp.29-41.

[23] Bahri M, Hettab A, Chaoui A, Kerkouche E. Transforming mobile UML statecharts models to nested nets models using graph grammars: An approach for modeling and analysis of mobile agent-based software systems. In *Proc. the 4th South-East European Workshop on Formal Methods*, November 2009, pp.33-39.

[24] Basso R. Wireless sensor networks in a vehicle environment [Master Thesis]. Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, 2009.

[25] B'Far R. Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML. Cambridge University Press, 2004.

[26] Carvalho G, Rodrigues R, Frances C, Costa J, Carvalho S. Modelling and performance evaluation of wireless networks. In *Proc. the 11th Int. Conference on Telecommunications*, August 2004, pp.595-600.

[27] Kumar B, Jasperneite J. Industrial communication protocol engineering using UML 2.0: A case study. In *Proc. the 7th IEEE International Workshop on Factory Communication Systems*, May 2008, pp.247-250.

[28] Kumar B, Jasperneite J. UML profiles for modeling real-time communication protocols. *Journal of Object Technology*, 2010, 9(2): 178-198.

[29] Lai A, Peng G, Tong H, Zhang G H, Bin H. Bluetooth host controller interface (HCI) using the Unified Modeling Language^{TM} (UML^{TM}). Technical Report, Institute of Communications Research, Singapore, 2002.

[30] Mahoney M, Elrad T. Distributing statecharts to handle pervasive crosscutting concerns. In *Proc. 2005 Workshop on Building Software for Pervasive Systems*, October 2005.

[31] Popovic M. Communication Protocol Engineering. CRC Press, 2006.

[32] Thramboulidis K, Mikroyannidis A. Using UML for the design of communication protocols: The TCP case study. In *Proc. the 11th International Conference on Software, Telecommunications, and Computer Networks*, October 2003.

[33] Barrera D. Communicating Systems with UML 2: Modeling and Analysis of Network Protocols. ISTE and John Wiley & Sons, 2011.

[34] Ali J, Tanaka J. Converting statecharts into Java code. In *Proc. the 4th World Conf. Integrated Design and Process Technology*, June 1999.

[35] Niaz I. Automatic code generation from UML class and statechart diagrams [Ph.D. Thesis]. University of Tsukuba, Japan, 2005.

[36] Niaz I, Tanaka J. Mapping UML statecharts to Java code. In *Proc. International Conf. Software Engineering*, February 2004, pp.111-116.

[37] Niaz I, Tanaka J. An object-oriented approach to generate Java code from UML statecharts. *International Journal of Computer & Information Science*, 2005, 6(2): 83-98.

[38] Tiella R, Villafiorita A, Tomasi S. FSMC+, a tool for the generation of Java code from statecharts. In *Proc. the 5th International Symp. Principles and Practice of Programming in Java*, September 2007, pp.93-102.

[39] Wagstaff K, Peters K, Scharenbroich L. From protocol specification to statechart to implementation. Jet Propulsion Laboratory Technical Report CL08-4014, 2008.

[40] Wasowski A. On efficient program synthesis from statecharts. In *Proc. ACM SIGPLAN Conf. Languages, Compilers, and Tools for Embedded Systems*, June 2003, pp.163-170.

[41] Boutekkouk F. Automatic SystemC code generation from UML models at early stages of systems on chip design. *Int. Journal of Computer Applications*, 2010, 8(6): 10-17.

[42] Kaliappan P, Koenig H, Kaliappan V. Designing and verifying communication protocols using model driven architecture and spin model checker. In *Proc. IEEE Int. Conf. Computer Science and Software Engineering*, December 2008, pp.13-19.

[43] Drusinsky D. Modeling and Verification Using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-Based Model Checking. Newnes, 2006.

[44] Amblard P, Lagnier F, Levy M. Finite state machines: Composition, verification, minimization: A case study. In *Proc. the 10th International Conference on Mixed Design*, June 2003.

[45] Prashanth C, Shet K. Verification of protocol design using UML — SMV. *World Academy of Science, Engineering and Technology*, 2009, 36: 544-548.

[46] Saleh K, Probert R. Automatic synthesis of protocol specifications from service specifications. In *Proc. the 10th IEEE International Phoenix Conference on Computers and Communications*, March 1991, pp.615-621.

[47] El-Fakih K, Yamaguchi H, Bochmann G, Higashino T. Petri net-based protocol synthesis with minimum communication costs. *Journal of the Franklin Institute*, 2006, 343(4-5): 501-520.

[48] Yamaguchi H, El-Fakih K, Bochmann G, Higashino T. Deriving protocol specification from service speci?cations written as predicate/transition-nets. *Computer Networks*, 2007, 51(1): 258-284.

[49] Bista B, Takahashi K, Shiratori N. Composition of service and protocol specifications. In *Proc. the 15th International Conference on Information Networking*, Feb. 2001, pp.171-178.

[50] Miles R, Hamilton K. Learning UML 2.0 (1st edition), O'Reilly Media, 2006.

[51] Ma J, Yu S. Practical rules for reduction on the number of states of a state diagram. In *Proc. Technology of Object-Oriented Languages Conference*, August 1998, pp.46-55.

[52] An Z, Peters D. Statecharts reduction and composition with properties. In *Proc. Newfoundland Electrical and Computer Engineering Conference*, November 2005.

[53] Rosen K. Handbook of Discrete and Combinatorial Mathematics. CRC Press, 2000.

Jehad Al Dallal received his Ph.D. degree in computer science from University of Alberta in Canada and was granted the best Ph.D. research award. He is currently working at Department of Information Science, Kuwait University as an associate professor. Dr. Al Dallal completed several research projects in the areas of software testing, software metrics, and communication protocols. In addition, he published more than 60 papers in conference proceedings and ACM, IEEE, IET, Elsevier, Wiley, and other journals. He also served as a technical committee member of several international conferences and an associate editor for several referred journals.

Kassem A. Saleh is currently a professor in information sciences at Kuwait University. He received his B.S., M.S., and Ph.D. degrees in computer science from the University of Ottawa in Canada. Dr. Saleh worked as a computer systems specialist at Mediatel, Bell Canada, from 1985 to 1991, and was on the faculty of Concordia University during 1991∼1992, Kuwait University from 1992 to 2000, and American University of Sharjah from 2000 to 2007. Dr. Saleh is also a Certified Information Systems Security Professional (CISSP). His research interests include software engineering, requirements engineering, and information security. Dr. Saleh has published more than 60 refereed journal papers and has presented numerous tutorials and lectures at international conferences and universities worldwide. Dr. Saleh recently published his first textbook on software engineering. Dr. Saleh is currently the editor-in-chief of *Journal of Software*.