A Novel Technique for Synthesizing Distributed and Concurrent Protocol Specifications

JEHAD AL DALLAL and KASSEM SALEH
Department of Information Sciences
Kuwait University
P.O. Box 5969, Safat 13060
KUWAIT

Abstract: - Several methods have been proposed for synthesizing computer communication protocol specifications starting from service specifications. Some protocol synthesis methods based on the Finite State Machine (FSM) model assume that primitives in the service specifications cannot be executed simultaneously. Others either handle only controlled primitive concurrency or have tight restrictions on the applicable FSM topologies. This paper proposes a concurrent-based protocol synthesis method that eliminates the restrictions imposed by the earlier methods. The synthesis method uses a sequential-based synthesis method to derive a sequential protocol specification (P-SPEC) from a service specification (S-SPEC) and then applies several transformation rules to re-model the resulting P-SPEC to consider the concurrency behavior specified in the S-SPEC.

Key-Words: - distributed applications, concurrent protocol, protocol specification, protocol synthesis, service specification

1 Introduction
A protocol can be defined as an agreement regarding the exchange of information between communicating entities. A full protocol definition defines a precise format for valid messages (syntax), procedure rules for the data exchange (grammar), and a vocabulary of valid messages that can be exchanged, with the meaning (semantics). In protocol design, interacting entities are constructed to provide a set of specified services for service users. While designing a communication protocol, semantic and syntactic errors may exist. Semantic design errors cause the provision of incorrect services to distributed protocol users. Syntactic design errors can cause the protocol to deadlock.

A communication system is most conveniently structured in layers. The Service Access Point (SAP) is the only place where a layer can communicate with its surrounding layers or service users. A layer can have several SAPs. The communication between the layer and its surroundings is performed using Service Primitives (SPs). The SP identifies the type of the event and the SAP at which it occurs. From the user’s viewpoint (high level of abstraction), the layer is a black box where only interactions with the user, identified by the SPs, are visible. The specification of the service provided by the layer is defined by the ordering of the visible SPs and is called Service Specification (S-SPEC). At a refined level of abstraction, the service provided by the layer is performed using a number of cooperating protocol entities. The number of protocol entities is equal to the number of SAPs available for the layer. The protocol entities exchange protocol messages through a communication medium. The protocol specification (P-SPEC) prescribes the exchange of messages between the protocol entities. Fig. 1 shows the two abstraction levels of a communication layer. Both S-SPEC and P-SPEC can be modeled using FSMs.

In autonomous communication systems, such as the Internet and mobile communication 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 is called message collision. If the protocol specification is not designed correctly, an unspecified reception error can occur (i.e., the protocol would be at a state where it is unable to handle the message that may arrive).

The construction of a protocol specification from a given service specification is called protocol synthesis. 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, ideally, provide the specified service. In addition, the syntactic correctness of the synthesized protocol is often a direct byproduct of the synthesis method. Several protocol synthesis methods have appeared in the literature, such as [1,2,3,4,5,6]. The methods introduced in [2, 5] are either not based on the FSM model or support only sequential applications. The method introduced in [3] supports only controlled non-sequential applications. The methods introduced in [4, 6] support non-sequential applications that have restrictions on the service specification model topology or on the allowed ordering of service primitives in the service specification.

This paper introduces a rule-based automatic synthesis method of protocol specifications not necessarily starting from sequential service specifications. The proposed synthesis method uses the method introduced in [1] to derive the sequential P-SPEC. Then, our method applies several transformation rules to re-model the sequential P-SPEC to consider the concurrency behavior specified in the S-SPEC. The synthesis method supports uncontrolled concurrent applications and is free of restrictions imposed on the service specification by the earlier methods. The introduced synthesis method uses an FSM-based model for modeling both service and protocol specifications. The paper is organized as follows. The related research is overviewed in Section 2. In Section 3, the models used for the service and protocol specifications are defined. In Section 4, the concurrent protocol synthesis method is introduced. Finally, Section 5 provides conclusions and discussion of future work.

2 Related Research
Two approaches are used in designing communication protocols: analysis and synthesis. In the analysis approach, a sequence of design, analysis, error detection, and correction is applied iteratively to produce error-free design. 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 start the derivation process from a complete service specification [1,2,3,4,5,6,7,8], and others do not [9, 10]. The protocol synthesis methods can be further classified according to the models used, which include finite state machines [1,3,4,5,6] and LOTOS-like [7, 8]. Some of the FSM-model-based service-oriented protocol synthesis methods consider the concurrency behavior of the protocol entities, including [3,4,6]. In [6], a concurrent timed protocol synthesis method is introduced, with three restrictions. The first restriction is that the S-SPEC model is assumed to be cyclic. The second restriction is that the required service is not allowed to have a specification that may cause a message collision. The last restriction is that if the S-SPEC has a state that has more than one outgoing transition, all the outgoing transitions from that state have to be associated with events that pass through the same SAP.

Kakuda et al. [4] introduced a concurrent protocol synthesis method, with two restrictions. The first restriction is that the S-SPEC model must be a tree. The second is that if a collision occurs between two protocol entities, the parallel path that has the higher priority proceeds, and the events that have not been executed in the other path yet are cancelled. This is called controlled concurrency. In controlled concurrency, if a message collision occurs, the problem is not solved by handling the collision in such a way that all parallel paths proceed without canceling some or all of their events.

Finally, in [3], a concurrent protocol synthesis method is introduced. The synthesis method extends the sequentially based synthesis method introduced by Saleh and Probert [1]. After applying the three steps introduced in [1], the extended method added transitions to solve the controlled concurrency problem, such that if a message collision occurs, either all parallel paths have to be re-executed or the parallel path that has the highest priority proceeds, and the events not yet executed in the other paths are cancelled. This solution is not practical, because it results in canceling or re-executing some events.

In this paper, the introduced synthesis method solves the true concurrency problem, so that if a message collision occurs, all parallel paths proceed without canceling or re-executing any event. In addition, the method introduced here eliminates the restrictions imposed by the methods introduced in [3] and [6].
and, therefore, is applicable to a wider range of applications.

3 Model Definition
Both the service and protocol specifications are modeled using FSM-based models. A FSM consists of states and transitions. It is limited to model sequentially based systems. This paper addresses synthesizing protocol specifications from a service specification that has concurrent behavior. Therefore, in this section, the FSM is extended to model the concurrent behaviors in the service specification. The extended model is called Extended Finite State Machine (EFSM). Each protocol entity is a sequentially based sub-system, and therefore, a traditional FSM can be used without extension to model the protocol entities. In this section, the models used are formally defined in the context of the layered communication system introduced in Section 1.

3.1 Service specification model
The service specification described in the EFSM defines sequences of primitives exchanged between users and processes through the service access points. A service specification has concurrent behavior if two or more service primitives pass through different SAPs simultaneously.

Definition 1: A service specification S-SPEC is modeled by an EFSM denoted by a tuple \((S, T, \sigma)\), where:
1. \(S\) is a non-empty finite set of service states. Each state \(s \in S\) is a choice, fork, joint, or leaf state. A choice state is a state that has one or more outgoing transitions, and only one transition is executed arbitrarily. A fork state, denoted by \(\|
\), is a state that has two or more outgoing transitions associated with service primitives that pass through different SAPs, and all these outgoing transitions are executed simultaneously. A joint state is a state that has two or more incoming transitions and one or more outgoing transitions. Finally, a leaf state is a state that does not have any outgoing transitions.
2. \(T\) is a finite set of transitions, such that each transition \(t \in T\) is a 3-tuple \(<\text{tail}(t), \text{head}(t), \text{SP}>\), where \(\text{tail}(t)\) and \(\text{head}(t)\) are, respectively, the tail and the head states of \(t\), 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 \(\text{SAP}(\text{SP})\).
3. \(\sigma \in S\) is the initial service state.

A path \(p\) in the ESFM is specified by the sequence of states \((s_1, s_2, \ldots, s_k)\) traversed by a set of successive transitions, where \(s_j = \text{successive}(s_i)p\). Parallel paths are specified by the sequence \((s_1, s_2, \ldots, s_k)\), where \(s_i\) is a fork state and \(s_k\) is either a joint or leaf state. Fig. 2 shows an S-SPEC example. In this example, paths \((s_2, s_3, s_4, s_7)\) and \((s_2, s_5, s_6, s_7)\) are parallel paths.

3.2 Protocol specification model
The protocol specification consists of the specifications of the protocol entities that cooperate to provide the service described in the service specification.

Definition 2: The protocol entity specification PE-SPEC\(_i\) is modeled by a FSM 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 \(s_{pi} \in S_{pi}\) is an image of one or more S-SPEC states. A state \(s_{pi}\) can be an image of more than one S-SPEC state if two or more S-SPEC states are combined during the protocol synthesis process. A path \(p_{pi}\) in the PE-SPEC\(_i\) is an image of path \(p_s\) in the S-SPEC if each state in \(p_{pi}\) is an image of one or more states in \(p_s\) and each state in \(p_{pi}\) is a pre-image of a state in \(p_s\). A parallel path in PE-SPEC is an image of a parallel path in S-SPEC.
2. \(T_{pi}\) is a finite set of transitions, such that each transition \(t_{pi} \in T_{pi}\) is a 3-tuple \(<\text{tail}(t), \text{head}(t), E_i>\), where \(\text{tail}(t)\) and \(\text{head}(t)\) are, respectively, the tail and the head states of \(t\), 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 \(PE_i\); or (3) an event message \(E\) received from \(PE_i\). The event of the second type is denoted by \(E_i\).
3. \(\sigma_{pi} \in S_{pi}\) is the initial protocol state.

Definition 3: In a protocol entity, the parallel paths that have the same image of fork state are denoted by \(R\), where \(R\) is written using the following BNF language rules:

\[
\begin{align*}
<R> & ::= [()|<E>|<C>^1..^k|]\r
<C> & ::= |<E>\r
<E> & ::= <E>| [()|<E>|<D>|]
\end{align*}
\]
<D>::=<operator> <E>
<operator> ::= .|+| ||
where E is a protocol event. Note that, in BNF, optional is denoted by [ ], and or is denoted by |. This formal language introduces three types of operators: ".", "+", and "||" to represent the operations before, or, and and, respectively. Each of these operators has two operands. The compound term A.B means that A has to be executed before B. The compound term A+B means that either A or B is to be executed. Finally, the compound term A||B means that both A and B have to be executed in any order. Fig. 3 shows the mapping between the compound terms and the represented FSM structure. The precedence of the operators is (1) ".", (2) "||", and (3) "+",. Using the formal language, the parallel paths in PE-SPEC1 given in Fig. 4 are represented by (?k2 || ?m3). Similarly, the parallel paths in PE-SPEC2 and PE-SPEC3 given in Fig. 4 are represented by ((?g2 . Q!/q2) || (?h3 . J!/j3)) and ((?g2 . Q!/q2) || (H!/h2 . M!/m1)).

Projecting the S-SPEC onto each SAP and applying a set of transition synthesis rules to the transitions of the projected S-SPECs to obtain the sequentially based PE-SPEC. Given the S-SPEC shown in Fig. 2, the resulting sequentially based PE-SPECs are shown in Fig. 4. In our extension, more states and transitions are added to the derived protocol entities to handle the concurrency behaviors.

To solve the concurrency problem, it is required to re-model each parallel path to contain all possible sequences of events considering the events in all other parallel paths. For example, either of the events ?k2 or ?m3 in PE-SPEC1 given in Fig. 4 can be executed first. When one of the events is executed, the protocol entity waits for the other event to be executed. Using Definition 3, clearly this means that the parallel paths have to be re-modeled from (?k2 || ?m3) to ((?k2 . ?m3)+(?m3 . ?k2)) as shown in Fig. 9. In other words, the re-modeling problem can be solved by (1) modeling the parallel paths using the language R given in Section 3, (2) applying a set of transformation laws to eliminate the presence of the "]" operator from R that represents the parallel paths, and (3) modeling the resulting R using FSM. Fig. 5 shows the synthesis algorithm.

### Synthesis algorithm:
**Derivation of concurrent protocol specification from a service specification**

**Input:** EFSM-based service specification  
**Output:** Concurrent FSM-based PE-SPECs

**Steps:**
1. Apply the synthesis method introduced in [2] on the S-SPEC to obtain the sequentially based PE-SPECs.
2. For each set of parallel paths in each PE-SPEC, do the following:
   2.1. Model the set of parallel paths using the language R defined in Definition 3.
   2.2. Apply the transformation laws detailed in Section 4.1 as needed until eliminating the presence of the "]" operator from R.
   2.3. Re-model the parallel paths in the FSM of the PE-SPEC using the resulting R.

Fig. 5: The synthesis algorithm

Having more than one outgoing transition from a state in a FSM means that one of these transitions can be executed. When having concurrent PE-SPECs, all such transitions have to be executed if the source state is an image of a fork state. Therefore, it is required to re-model the PE-SPEC to show that all these transitions can be executed. In Step 2.1 of the synthesis algorithm, the parallel paths are modeled using the language R. Using this
language, three possible operators are used, including "||". This operator is not modeled in typical FSM. Therefore, a set of transformation laws are introduced to eliminate the "||" operator from R that models that parallel path. The laws are listed below and demonstrated in Fig. 6 and Fig. 7. Note that in these transformation laws, un-bold letters represent events, and bold letters represent sequence of events separated by the "." operator.

**Distributive Law 1**: \( A||B = A.B + B.A \)

**Distributive Law 2**: \( (A.B) || (C.D) = (A.(B|| (C. D)))+ (C.(D|| (A.B))) \)

**Distributive Result 1**: \( A || (C.D) = (A.C.D) + (C.(A||D)) \)

**Commutative Law 1**: \( A||B = B||A \)

**Commutative Law 2**: \( A+B = B+A \)

**Commutative Law 3**: \( A.B \neq B.A \)

**Associative Law 1**: \( (A||B)||C = A||(B||C) \)

**Associative Law 2**: \( (A+B)+C = A+(B+C) \)

**Associative Law 3**: \( (A.B).C = A.(B.C) \)

Fig. 6: The distribution laws

Fig. 7: The commutative laws

**Fig. 8**: Applying the transformation laws on R for each of the parallel paths given in Fig. 4.

For parallel paths in PE-SPEC_1
\[ R = (?k_2||?m_3) = (?(k_2 . ?m_3)+( ?m_3 . ?k_2)) \]

For parallel paths in PE-SPEC_2, assuming A:G/!g_3, B:?q_3, C:K/!k_1, D:?h_3, E:J/!j_3
\[ R = (A.B||C||D.E) \]

For parallel paths in PE-SPEC_3, assuming A:H/!h_2, B:?j_2, C:M/?m_1, D:?g_2, E:Q/?q_2
\[ R = (D.E||A.B.C) \]

This R is similar to the one obtained in the above second step for PE-SPEC_2. Therefore, it has the same final result. Note that the labels of A, B, C, D, and E for PE-SPEC_3 are different than the ones for PE-SPEC_2.

**Fig. 8**: Applying the transformation laws on R for the parallel paths given in Fig. 4.

After applying the transformation laws, the resulting R is modeled using states and transitions straightforwardly using Definition 3. Fig. 9 shows the resulting PE-SPECs for the sample example used in this paper.

### 4 Conclusions and Future Work

In this paper, a synthesis method for concurrent protocol specifications from service specifications is introduced. Both the service and protocol specifications are modeled using FSM-based models. The service specification FSM-based model
is extended to model concurrency behaviors. The synthesis method first uses a previously introduced method to synthesize sequentially based protocol specifications. Then, the synthesis method applies several transformation rules to re-model the derived protocol specifications. The resulting FSM representation for the concurrent PE-SPECs can be minimized to eliminate redundant states and transitions [11]. In contrast with methods introduced earlier, the synthesis method introduced here solves the true concurrency problem, such that if a message collision occurs, all parallel paths proceed without canceling any event. In addition, the synthesis method introduced here eliminates all the restrictions imposed by the earlier methods. As a result, the synthesis method is applicable to a wider range of concurrent applications.

![Fig. 9: The resulting PE-SPECs that consider the concurrency behaviors specified in the S-SPEC given in Fig. 2](image)

The synthesis method introduced in this paper does not consider medium channel delays and timing requirements that can be provided in the service specification. These two issues are subjects for future work.

**Acknowledgement**

The author would like to acknowledge the support of this work by Kuwait University Research Grant WI03/05.

**References:**


