Communication Abstraction and Verification in Distributed Scenario Integration*

Aziz Salah[1], Rabeb Mizouni[2], Rachida Dssouli[3]

[1]Department of C.S.
University of Quebec at Montreal
Email: salah.aziz@uqam.ca
[2]Electrical & Computer Engineering
Concordia University
Email: mizouni@ece.concordia.ca
[3]Institute For Information System Engineering, Concordia University
Email: dssouli@ciise.concordia.ca

Abstract

Successfully modeling and analysing requirements are of the main challenges to face up when it is time to produce a formal specification for distributed systems. Scenario approaches are proposed as an alternative to make this process easier. They are based on the decomposition of the system behavior into intuitive pieces that are distributed scenarios. In this paper, we propose an approach to detect the unspecified reception errors by the integration of scenarios. The decision about such property is made possible according to an architectural communication model which states the communication abstraction level we are considering. We synthesize from message sequence char a set of automata, one per object. Then, we give decision procedure for unspecified reception faults.

Keywords: Communication Abstraction, MSCs interpretation, Scenarios, Verification, Unspecified Reception.

1. Introduction

The distributed system community agrees that choosing a specification language with the appropriate abstraction communication level is crucial for the targeted verification properties like unspecified reception, service denial, and deadlock free. Moreover, the abstraction makes the understanding and the development of distributed systems easier except that some properties cannot be checked when the level of abstraction is not suitable. Consequently, deciding about the adequate level of abstraction is a real challenge to face up when choosing a specification language.

On the other hand, scenario approaches have emerged to facilitate the construction of a formal specification by promoting a "Divide and Conquer" strategy. They consist of composing scenarios, where each one describes a part of the system behavior, into a global and formal specification. The latter is represented by a set of automata which

* Supported by NSERC

should conform to the original scenarios and be checked for some desired user specified properties.

In this paper, we report a practical experience emphasizing the importance of the abstraction communication level in scenario approaches and verification. We were confronted to this fact when we had to choose semantics for message sequence charts (MSCs) [5]. In previous work [9], synchronous semantics of MSCs were chosen to interpret messaging order between objects. With this semantics, we could check for deadlocks, but we were unable to verify that an object does not send a message to another one when the latter is not ready to receive it. This is known in the communication community as the unspecified reception fault. A synchronous communication model is inappropriate for checking the unspecified reception fault because the abstraction assumes a hand shaking before the communication. Hence, we adopt another interpretation of MSCs based on an asynchronous communication model. In our approach, a system specification is given as a set of MSCs from which we synthesize an automaton for each object. Those automata are composed and checked to be unspecified reception fault free. In what follows, the term verification is used to mean model checking.

The paper is structured as follows: in Section 2, we give an overview of the semantics we are using. Section 3 presents the annotation of MSCs, and the synthesis of the overall system automaton and its verification against the unspecified reception fault. In Section 4, we propose a discussion about different level of abstraction and how they affect the verification of system properties. Finally, Section 5 closes the paper with conclusions.

2. Preliminaries, definitions and formal semantics

Message Sequence Charts (*MSCs)* [5] are a commonly used visual representation for scenarios of interaction among objects. An MSC focuses on message exchange and shows a partial order of events. A message represents an interaction between two objects, a sender and a receiver. MSCs are less intuitive than expected as they may display an order of events which is not always the only case supported by an asynchronous communication based implementation.

In this paper we assume that the distributed system in development is composed of set of objects $\Omega = \{O_1, O_2, \ldots, O_n\}$ and their environment *Env*. The formalization of MSCs allows the definition of the real partial order according to particular architectural communication assumptions. The formalization of MSC was treated by many researchers [2, 4], and we follow a similar approach.

Definition 1. *An MSC is a structure* $M=(I, SE, RE, r, L, p, <_D, <_m)$ *where*

- $I \subset \Omega \cup \{Env\}$ *is a set of objects;*
- *SE is the set of sending events and RE the set of receiving events. We denote by* SE_O *(respectively* RE_O*) the set of sending events (respectively the set of receiving events) in object O;*
- $r : SE \rightarrow RE$ *maps a sending event to its receiving event. r is a bijection.;*

- *L is a set of labels of the messages in the MSC;*
- *p : $SE \cup RE \rightarrow I$ maps a sending event or receiving event to an object from I;*
- *$<_D = \cup_{O \in I} <_O$ where $<_O \subset SE_O \cup RE_O \times SE_O \cup RE_O$ is a total order between the local events in object O according to the visual order as displayed in the MSC;*
- *$<_m = \{(s, r(s)) \mid s \in SE\}$ is an ordering relation which means that a message cannot be received before it is sent.*

The previous definition is very general and does not include any assumption about the communication architecture in the system yet. The order $<_D$ *capture the order as displayed* in the MSC and it may be not granted by the semantics of MSCs for a particular communication architecture. As the behavior described by MSCs will be translated into a set of automata, their respective semantics should be compatible. In our approach, since the communication between objects uses no buffering facility, we assume that the FIFO order is preserved when an object receives two or more messages from the same object. Thus, the events should fulfill the partial ordering relation $<_{FIFO} = \{(r(s),r(s')) \mid (s,s') \in <_D$ *and* $p(s)=p(s')$ *and* $p(r(s))=p(r(s'))$ *and* $s \in SE$ *and* $s' \in SE\}$. Furthermore, automata modelize autonomous objects. Consequently, an object has the control over its sending events. Hence, its scheduling of sending events is granted according to the visual order $<_D$. We also grant the local visual order between a receiving event and the next sending events in an object. Those architectural assumptions are expressed by the following control ordering relation : $<_C = \{(e,s) \mid e \in SE \cup RE, s \in SE$ *and* $(e,s) \in <_O$ *and* $O \in I\}$. Finally, the interpretation of an MSC is given by the partial order relation $<$ defined as the transitive closure of the combination of the three partial orderings $<_C$, $<_m$ and $<_{FIFO}$:

$$< = (<_C \cup <_m \cup <_{FIFO})^*$$

The behavior of an object O_i is described by an automaton A_{O_i} defined by:

Definition 2. The automaton of object O_i *is a structure* $A_{O_i} = (S_{O_i}, s_{O_i init}, T_{O_i}, E_{O_i})$ where S_{O_i} denotes its set of states, $s_{O_i init}$ its initial state, and $T_{O_i} \subset S_{O_i} \times E_{O_i} \times S_{O_i}$ its set of transitions. The set of events E_{O_i} is partitioned into two components SE_{O_i} and RE_{O_i} that are respectively the message reception and sending event sets.

Objects communicate by exchanging messages. In this work, the automata of objects are assumed to communicate according to the semantics of input/output automata defined in [7]. The event of sending of a message are labels like $(O_i,O_j.m)$ where O_i and O_j are objects of the system. $(O_i,O_j.m)$ means that message m is sent from O_i to O_j.

Definition 3: An automaton $A_{O_i}=(S_{O_i},S_{O_i init},T_{O_i},E_{O_i})$ is said to be reception completely specified automaton (RCSA) iff $\forall O \in \Omega \cup \{Env\} \setminus \{O_i\}, \forall se \in SE_O, \ \forall s \in S_{O_i}$ $\exists (s, r(se), s') \in T_{O_i}$.

An automaton is RCSA means it accepts the reception of any message in any state. Objects are autonomous, so they control sending messages, but not receiving them. Usually objects automata are not RCSA. Consequently, if a message is sent to an

object while its current state has no transition for the reception of that message, the system falls into an unspecified reception fault. We formalize the communication between RCSA automata by their parallel composition:

Definition 4. If $A_{O_1}, A_{O_2}, \ldots$ and A_{O_n} are RCSA then their parallel composition denoted by $\prod A_{O_i}=(S,s_{init},T,\Sigma)$ is an automata where:

- $S \subset S_1 \times S_2 \times \ldots \times S_n$ and $s_{init}=(s_{O_1 init}, s_{O_2 init}, \ldots, s_{O_n init})$,
- $T \subset S \times \Sigma \times S$ is its set of transitions defined by the following rules:
  - Rule 1 : $(s_i,a,s_i') \in T_{O_i}$ *and* $(a=(O,O_i.m)$ *or* $a=(O_i,O.m))$ *and* $O \in \{O_i, Env\}$ *implies* $((s_1,..,s_{i-1},s_i,s_{i+1},..,s_n),a, (s_1,..,s_{i-1},s_i', s_{i+1},..,s_n)) \in T$
  - Rule 2: *let* $O_i \neq O_j \in \Omega$: $(s_i,a,s_i') \in T_{Oi}$ *and* $a=(O_i,O_j.m)$ ***implies*** $(s_j,a,s_j') \in T_{Oj}$ *and* $((s_1,..,s_i,..,s_j, ..,s_n),m, (s_1,..,s_i',..,s_j',..,s_n)) \in T$

Rule 1 treats internal actions and communication with the environment while Rule2 treats communication among two different objects *Oi* and *Oj*.

3. Detection of unspecified reception in annotated MSCs

Our goal is to synthesize a single automaton for each object from the given MSCs. Let $A_O(M)$ be an automaton implementing the behaviour of an object O in an MSC M. The synthesis $A_O(M)$ is based on local cuts [3]. A local cut C is set of events from E_O such that if e in C and there exists e' in E_O and $e'<e$ then e' is in C. For each local cut C of E_O such that $C \neq E_O$, there exists e in E_O such that $C \cup \{e\}$ is a local cut of E_O, ie., C can be extended with an event and still remains a cut. The cuts represent the states of the automaton $A_O(M)$. The empty cut is *its* initial state, and E_O is its terminal state. The transition of $A_O(M)$ *are* $(C,e, C \cup \{e\})$ such that C and $C \cup \{e\}$ are local cuts of E_O.

For an object, we aim at connecting together its automata coming from different MSCs. The procedure consists of merging states from those different automata in order to synthesize an automaton implementing the behavior of the object in all the given MSCs. Identifying which states to merge is based on annotating MSCs with state information. The annotation allows not only capturing infinite traces by introducing cycles, but also recognizing shared states in different scenarios and thus determining relationships between them. In addition, annotating the MSCs gives the analysts the opportunity to add their interpretations regarding the states of an object.

High level MSCs (HMSCs) are MSC-graphs where each node is an MSC [5]. They provide a means to define how MSCs can be composed. They can be used jointly with the state annotation we are proposing. However, in our case state annotation encloses already the expressiveness power of HMSCs.

At this stage, each object has an automaton that represents exactly its behavior as specified by the user. These automata are usually not RCSA. We propose to complete each of their state by transitions that make them RCSA in order to enable their composition. Each state is completed by transitions to receive any of the possible automaton received messages. Those added transitions end up in a special state called *UR*. It denotes the fact that the reception of a message in a state where it was not

expected to be received in is an unspecified reception fault. The completely specified automata of the different objects are then combined according to the parallel composition. The composition automaton includes thus a state where one of its components is *UR,* but if such a state is reachable, it shows that the unspecified reception fault is possible. The scenario of the fault is given by the path leading to that state. Thanks to traceability between the MSCs and their automata, the origin of the unspecified reception fault can be retrieved. The fault can be recovered by changing either the MSC state annotation or the MSCs themselves.

4. Communication abstraction level vs. verification in scenario approaches

Model abstraction level can be an obstacle to the verification of certain properties in the distributed systems. In fact, if that level is too low, the verification problem becomes complex, even undecidable. However, if the level is too high, the model becomes inappropriate for certain verifications.

The semantics of MSCs can be divided into synchronous and asynchronous models. We believe that the first one is the more abstract. It makes stronger assumption as it uses "Rendez-vous" based communication model. Furthermore, synchronous interpretation of MSCs makes them more intuitive because the local visual order for each process is granted.

However, to describe autonomous communicating object behavior, the asynchronous communication model allows a decoupled relation between the sender and the receiver objects. Different types of asynchronous semantics are depicted according to whether or not buffering capabilities are used. Nevertheless, when buffering is supported, two characteristics have to be specified: the queuing is considered to be FIFO or not, and the buffering is done per source, or per object [3]. For all the previously described semantics of MSCs, the verification of properties falls into a reachability analysis of a specific finite composition automaton.

As stated earlier, the sequencing of MSCs can be specified by an HMSC. The interpretation of an HMSC is given by extending the ordering of its MSCs. There are two possible extensions: the first one assumes that the system has to observe all the events of an MSC before moving forward into another one according to the HMSC graph; i.e. all objects synchronize before the system continues with another MSC. That is what Alur and Yannakakis [3] called synchronous semantics of HMCS. In contrast, this HMSC semantics is not more expressive than MSC with the state annotation we are using. The second interpretation of HMSC consists of extending the MSCs semantics with enabling the system to freely execute events from different MSCs provided that for each object, its local events are completely executed before moving forward into another MSC. This semantics renders an HMSC difficult to understand. The verification of synchronous HMSC is decidable while it is not for asynchronous ones [3, 8].

It is worth to point out that we cannot talk about the verification of an abstract model before checking its realizability. A system is realizable if there exists an automaton supporting the same semantics as the abstract specification. Moreover, it

safe is realizable if there exists a deadlock free automaton with the same semantics. It has been proved that both realizability and safe realizability of MSCs under any of the above defined abstraction level are always decidable [1]. Nevertheless, in general, both kind of realizability of HMCSs are undecidable, but it has been proved decidable for some specific kinds of HMSC[6, 8]

5. Conclusion

We have so far presented an automatic approach for detecting unspecified reception fault after constructing the overall system automaton. The specification of the system is given as a set of MSCs. Adopting a synchronous interpretation for MSCs instead of asynchronous one, in this case, is inappropriate as this level of communication abstraction is not concerned by such fault.

It is clear that the communication abstraction level is a determinant factor in the verification of distributed systems against some desired properties, especially in the case of scenario based approaches. The choice should be dictated by the properties to be verified. In fact, by practicing communication abstraction, some properties may be lost but the verification becomes consequently easier to achieve. However, a tradeoff should be made so that the abstraction model preserves the desired properties

References

[1] R. Alur, K. Etessami, and M. Yannakakis, "Inference of message sequence charts," presented at 22nd International Conference on Software Engineering, 2000.

[2] R. Alur, G. Holzmann, and D. Peled, "An Analyzer for Message Sequence Charts," *Software: Concepts and Tools*, vol. 17, pp. 70-77, 1996.

[3] R. Alur and M. Yannakakis, "Model checking of message sequence charts," presented at Proceedings of CONCUR'99: 10th International Conference on Concurrency Theory, 1999.

[4] H. Ben-Abdallah and S. Leue, "Syntactic Detection of Process Divergence and Non-local Choice inMessage Sequence Charts," in *TACAS*, 1997, pp. 259-274.

[5] ITU, *Recommendation Z.120: Message Sequence Chart (MSC)*, 1999,.

[6] M. Lohrey, "Safe realizability of high-level message sequence charts," presented at Proceedings of 13th International Conference on Concurrency Theory - CONCUR '02, 2002.

[7] N. Lynch, "I/O Automata: A model for discrete event systems," presented at 22nd Annual Conference on Information Sciences and Systems, Princeton University, Princeton, N.J., 1988.

[8] A. Muscholl and D. Peled, " Message sequence graphs and decision problems on Mazurkiewicz traces," presented at MFCS'99, 1999.

[9] A. Salah, "A use case driven synthesis of state diagrams," presented at Eighth Maghrebian Conference on Software Engineering and Artificial Intelligence, Tunisia, 2004.