분산 객체지향 시스템을 위한 정형 명세 방법에 관한 연구

이 상 범†

요 약

분산처리 시스템은 기존의 순차시스템과의 많은 차이점으로 인해 개발 과정에서 사용자 요구사항을 정형적으로 기술하고 검증에 도와줄 모델링 기법과 방법의 필요성이 증대하고 있다. 분산객체시스템에 있어 커뮤니케이션의 역할이 중요함에도 불구하고 일반적으로 설계과정에서 통신에 관한 것에 대하여 자세히 정의하는 것이 중요시되지 않았다 하지만 분산시스템인 경우 비분산시스템에 비해 통신 형태를 결정하는 것이 중요하기 때문에, 이에 대한 것을 설계 단계에서 결정하고 정의하는 것이 중요하다 따라서 본 논문에서는 분산시스템에 있어 temporal logic과 Petri net를 이용한 정형 명세 기법을 소개하고 있다. Temporal logic에 의한 명세 기법은 정의된 기본 predicates를 이용한 룰을 이용하여 여러 형태의 통신 형태를 정의하며 Petri net방법은 패트리넷의 특성을 이용한 동적인 행위를 표현하는데 도움을 가져다준다.

Formal Specification Methods for Distributed Object-Oriented Systems

Sang-Bum Lee†

ABSTRACT

As distributed computing systems become popular, many modeling techniques and methods have been developed to specify the specification formally and verify the distributed/concurrent systems In spite of importance of communication in distributed object-oriented systems, specifying of communication method generally has not been emphasized in the design phase. One reason is due to the system designer misunderstanding, that is, a specification needs to be independent on the implementation However, since defining communication pattern in distributed object-oriented systems is more serious than the required message passing method in the design phase, specifying the communication pattern is necessary instead of postponing until the implementation.

In this paper, two formal specification techniques, temporal logic method and Petri nets method, for the communication in distributed systems are discussed. One is based on the temporal logic, which specifies the different patterns of primitive predicates This method enables to define the underlying mechanism which can be interpreted as constraints. The Petri net method helps to specify the dynamic behavior of communicational patterns using the properties of Petri nets.

1. Introduction

Distributed systems which consist of a set of independent parallel-executable modules tend to support the message passing for parallelism instead of having a shared global memory. Since object-oriented systems and distributed systems share similar properties, the combination of these systems

※ 이 연구는 단국대학교 대학연구지원비의 지원으로 연구되었음
† 정 회 원 단국대학교 전자계산학과 교수
논문접수 1998년 10월 28일, 심사완료 · 1999년 5월 30일

is somewhat natural [1]. An object in an object-oriented system inherently has a suitable form for a distributed systems [2]. Therefore, the distributed object-oriented systems have been developed by applying object-oriented techniques to distributed systems

Meanwhile the object-oriented approaches to software development have received an increased emphasis since the early 1980's [3]. These new software development techniques are expected to be used widely due to powerful features such as information hiding, modularity, abstraction, and localization [4]. Systems designed from these approaches consist of a set of object modules and the structure of the system tends to be flat instead of hierarchical structure. One of the main features in object-oriented systems is message passing, i.e., a set of objects communicates with each other by sending or receiving messages. An object begins to be activated when it receives a message from other object.

Most distributed object-oriented programming languages support parallelism between objects, but a few languages support inter-object concurrence. Basically there are two different types of message passing: synchronous message passing and asynchronous message passing The message passing method of systems is mainly dependent on the communication mechanism of implementation programming language. While CSP [5] supports synchronous communication, ABCL/1 [2], [6] supports asynchronous message passing. Specifying of the message passing method in the design phase has been ignored because the system designers believe that a specification needs to be independent on the implementation However, since defining communication patterns in distributed systems is more serious than non-distributed systems, it is desirable to decide and specify the required message passing method in the design phase instead of postponing until the implementation phase. By doing this, an appropriate programming language can be selected and the system can be implemented properly without losing the requirements of the system designer. In this paper, two specification techniques particularly for the message passing in distributed object-oriented system development are introduced. The outline of this paper is as follows. In Section 2, the background of this work is introduced The specification methods of message passing methods are discussed in Section 3. Section 4 contains the conclusion

2. Backgrounds

2.1 Formal specification languages

Compared to other system development, a few specification languages are developed to represent the specification of desired systems according object-oriented approaches. A specification written by a formal specification language helps to prevent different people from interpreting the system differently. It enables to remove the ambiguity in the specification. However, there is an overhead to learn formal specification languages since these typed languages are developed based on the mathematical theory and contain mathematical notations in their syntax. Use of formal specification languages in software development has increased due to their benefits. We have defined a formal specification language, called DOSL(Distributed Object-based Specification Language), particularly for distributed object-oriented system development Basically, DOSL adopts some features form CSP, ABCL/1 and temporal logic expression [7]. One of prominent features in DOSL is its powerful message passing statements which can express various communication patterns explicitly The detailed features of this language are introduced in [8] and [9].

2.2 Temporal logic and operators

Temporal Logic(TL), defined by Pnueli [7], is a branch of the propositional logic and contains temporal operations which helps simply to represent the logical relationships among a class of time varying events and also infers such relationship from them.

TL can be viewed also as a special case of Predicate Logic or First Order Logic (FOL). The proof technique for temporal logic is completely decidable while that for FOL is only partially decidable

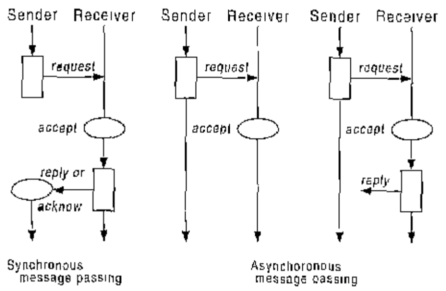
The advantages of using TL can be summarized as follows. 1) The inference techniques for TL are more powerful than that for FOL. 2) It has more expressive or modeling power than that of the Propositional Logic. 3) It has a completely decidable inferencing method, like that of PL.

There exist many different types of TL that have studied in mathematical logic and which have applications in computer science. The various forms of temporal logic differ from each other in two aspects: the type of temporal operators or relationships that are allowed in the formulas and the nature of underlying time scale.

Temporal logic formulas are defined by combining temporal operators, logical operators and logical expressions. Use of systems is introduced in [10] and [11]. The basic temporal operators are as defined as follows.

- □ :: always true in the future
- ○ :: *the next state is true*
- ◇ :: sometimes or eventually true in the future

These temporal operators are used to denote the explicit communication patterns in distributed object-oriented systems according to their semantics.



(Fig. 1) The two message passing methods

3. Specification Methods of Message Passing

There are two different methods of communication in distributed systems: synchronous and asynchronous These two patterns are introduced pictorially in Figure 1 In DOSL statements for message sending and message accepting are defined explicitly by attaching a temporal operator as a prefix, i. e., the temporal operator precedes the message passing statement to specify the specific communication method We assume that the message passing pattern is determined by attaching a temporal operator in front of the message sending statement instead of specifying the same operator to the corresponding message accepting statement again. In Table 1, modified message passing statements in DOSL are described. The full syntax and formal semantics of DOSL are defined in [9]

While the two operators, □ and ◇, are used to denote asynchronous message passing, and operator ○ is used to represent synchronous message passing according to their meaning. The operator □, which means that the following statement is always true, is used to denote asynchronous message passing in which a sender object does not receive any information back from a receiver object, i.e., it continues execution immediately after *sending* a message. A temporal operator ◇ is used to represent asynchronous message passing in which a sender object *will receive a requested information* eventually but it does not suspend its execution during meantime. Another operator ○ is used to *denote synchronous communication in which* a sender object has to wait until a receiver object gets the message.

In fact, there is no direct relationship between temporal operators and message passing methods but we *match each other according to their* meaning. To complement this, two specification methods which enable to represent the concept of time are introduced in the following subsections to specify formally the message passing methods. In addition, these following methods can be used to define the semantics of DOSL.

〈Table 1〉 Message passing statements in DOSL

- message sending statements
  - ○ *(send x to Obj)*
    : synchronous message passing, receiving an acknowledgment
  - ○ *(send x to Obj & get y)*
    : synchronous message passing, receiving a return message
  - □ *(send x to Obj)*
    : asynchronous message passing without a return message
  - ◇ *(send x to Obj & get y)*
    : asynchronous message passing, receiving a return message
- message accepting statement
  (accept x)

note: x is a set of parameters and Obj is an object
y is a future variable where the return information be saved

3.1 Temporal logic specification

In a logic specification, first of all, a set of elementary predicates which represents relevant properties about the states or the events of the system needs to be defined. Based on these predicates, a set of rules is introduced as a formal predicates followed by an arrow which represents the implication and followed by consequence of the predicates. Detailed explanation of this approach can be found in [9]. With these rules the behavior of a system can be formally specified Here, we use the temporal logic expression for specifying of communication methods since it has the power of expressing the lapse of time. Temporal logic specification method [10] has received an attraction as a technique to specify a system whose behavior is relate to the lapse of time, such as real-time systems

The underlying meaning (semantics) of the message passing statements, the specification of communication, are defined using a temporal logic specification method. One of the important features is its explicit expressiveness of the communication patterns in the message passing statements To specify different typed communication patterns, a set of the predicates is defined in Table 2. Derivation of the set of rules for specifying each message passing method is introduced in the following subsections.

〈Table 2〉 A set of predicates for the logic specification

- *send(O1, O2, msg, t)*
  an object *O1* sends a message *msg* to another object *O2* at the global time *t*
- *receive(O1, O2, msg, t)*
  an object *O1* receives *msg* from another object *O2* at the global time *t*
- *suspend(O1)*
  an object *O1* is in a suspended state

note msg can be replaced by ack or reply which stand for an acknowledgment and the required information, respectively

3.1.1 Synchronous message passing

In synchronous message passing, an object which sends a message to a particular object suspends until its partner object sends back a message to it, i.e., the receiver object has to send back an acknowledgment or a requested information to the sender object to ensure that it has received a message. The sender object can be active again after it receives an acknowledgment or the required information. This way of communication may meet a deadlock situation when two objects send messages to each other simultaneously. Moreover, it does not fully support the potential parallelism because an object has to suspend after sending a message until it receives a message [12]. Rules for synchronous message passing are defined as follows

- $send(O1,O2,msg,t') \rightarrow \Diamond\ receive(O2,O1,msg,t) \cap suspend(O1) \cap t' > t$
  remark) When an object *O1* sends a message *msg* to another object *O2*, *O2* will receive it at time *t'* an *O1* suspends for meantime.

- $receive(O1,\_,msg,t) \cap receive(O1,\_,msg',t) \rightarrow msg = msg'$
  remark) If *O1* receives two messages at the same time, these two should be the same messages, i.e., *O1* cannot accept two different messages simultaneously

- *receive(O1,O2,msg,t)* $\rightarrow$ $\Diamond$*(send(O1,O2,ack,t')* $\cup$ *send(O,O2,reply,t'))* $\cap$ $t' > t$

  remark) If *O1* receives a message from *O2*, then *O1* will eventually send back an acknowledgment or the requested information to *O2*.

- *suspend(O1)* $\cap$ *((receive (O1,_,ack,_)* $\cup$ *(receive (O1,_,reply,_))* -> ┐*suspend(O1))*

  remark) If *O1* is in a suspended state, it resumes execution after receiving an acknowledgment or a reply from another object. i.e., before it receives a message it remains suspended.

- *send(O1,O2,ack,t)* -> ┐*(send(O1,O2,reply,t))*

  remark) *ack* and *rely* cannot be sent simultaneously.

3.1.2 Asynchronous message passing method

The sender object, in asynchronous message passing, continues its execution without waiting the receiver object to receive a message but there are two different cases: The sender does not receive any message from the receiver object and the sender object will eventually receive the requested message. The internal behavior of these two are not the same as those of synchronous message passing. Here we explain each case separately.

■ In case of having no return message

- *send(O1,O2,msg,t)* $\rightarrow$ $\Diamond$ *receive(O2,O1,msg,t')* $\cap$ ┐*(suspend(O1))* $\cap$ $t' > t$

  remark) When *O1* send a message to *O2* at time *t*, *O2* eventually receives a message at the time *t'* but *O1* will not be suspended.

- *receive(O1,O2,msg,t)* $\cap$ *receive(O1,O2,msg',t)* $\rightarrow$ *msg = msg'*

  remark) Two different messages cannot be accepted at the same time

■ In case of having a return message

- *send(O1,O2,msg,t)* $\rightarrow$ $\Diamond$ *receive(O2,O1,msg,t')* $\cap$ ┐*suspend(O1))*

  remark) When *O1* sends a message to *O2* at time *t*, *O2* eventually receives the message at time *t'* and *O1* will not be suspended.

- *receive(O1,O2,msg,t)* $\cap$ *receive(O1,O2,msg',t)* $\rightarrow$ *msg = msg'*

  remark) Two different messages cannot be accepted at the same time

- *receive(O1,O2,msg,t)* $\rightarrow$ $\Diamond$*(send(O1,O2,reply,t'))* $\cap$ $t'=t+n$

  remark) When *O1* receives a message *msg* from *O2* at the time *t*, *O1* has to send back the requested message within *n* time units

Note: If a predicate in the right-hand side of the rule does not contain any temporal operator, it is assumed that an operator $\Box$ is attached as a prefix.

3.2 Petri nets specification method

Petri nets, designed by C.A. Petri [13], have been widely used as tools for the design of communication protocols [14] and distributed/concurrent computing systems [15]. The power of modeling a system with Petri nets has been increased by extension to the original Petri net model There are two approaches to use of Petri nets in software development. One approach is to view the Petri net model as an analysis tool where the system properties are analyzed and modeled in Petri nets which then analyzed for such properties as safeness, boundness, liveliness, and readability A second use of the Petri nets in the specification and design is to use them for the entire specification and design process, thus requiring the transformation of Petri net representations into systems. A Petri net is defined as follows but the detailed explanation of the Petri nets is not included in this paper.

Definition: A Petri net is a triple

$$N = (S,T,F)$$

where

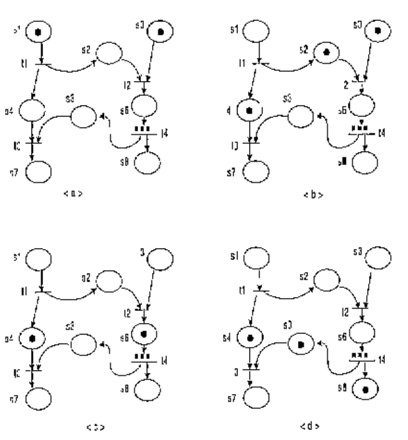
i) *S* and *T* are disjoint set of places and transitions, respectively.

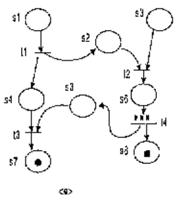
ii) $F \cup (S\times T) \cup (T\times S)$ is a relation between places and transitions.

We assume that s_i and t_j represent the elements of two sets, S and T. In the net, S, T and F are represented by circles, bars and arcs, respectively.

Using the properties of Petri nets, the dynamic behavior of message passing can be specified. Compared to the previous method, it shows the dynamic behavior pictorially and easily to be understood. We assume that two objects communicate each other by passing messages. O1 and O2 represent a sender object and a receiver object, respectively. In addition, we assume that initially two objects are ready to communicate.

The meaning of synchronous message passing is represented by a set of Petri nets and given in Figure 3.1. Assume the two objects communicate each other, the left half of the net represents the sender object O1 and the right half represents the receiver object O2. The sequence of movement of tokens in Petri nets is explained as follows: Initially, when O1 and O2 are ready to communicate, the tokens are placed in the two places, t1 and t3, as <a>. (The place having a token implies the current execution state). After O1 sends a message, it becomes an suspended state (s4) and O2 accepts the message (t2 fires) and goes to the next state (s6) as <b> and <c>. Meanwhile O1 waits until it receives back the information or an acknowledgment: t3 can only fire when the two places s4 and s5 have the tokens, that is, when O2 sends back a message to O1, O1 resumes the execution by firing t3 as <d> and <c>

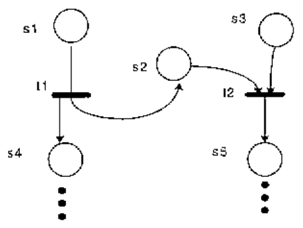




s1 a message is in the queue of the sender object O1
s2 a message is in the queue of the sender object O2
s3 O2 is ready to receive a message
s4 O1 is in the suspended state
s5 a returned message is in the queue of O1
s6 O2 is ready to execute by interpreting the message
s7 O1 is ready to execute some statement
s8 O2 is ready to execute some statement
t1 O1 sends a message to O2
t2 O2 receives a message
t3 O1 receives back a message from O2
t4 O2 sends back a message to O1

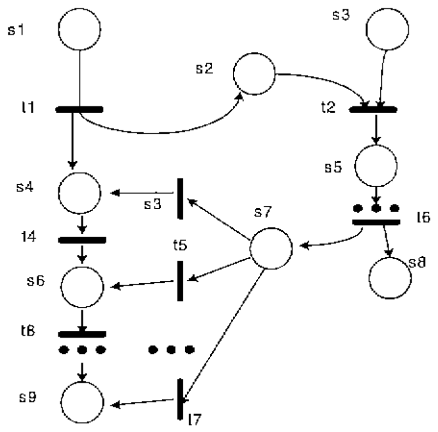
(Fig. 3.1) The synchronous message passing

Asynchronous message passing which does not receive back any information from the receiver object is represented with a Petri net in Figure 3.2. Initially the places s1 and s3 have tokens which mean that the two objects are ready to communicate. When O1 sends a message to O2, the transition t1 fires and consequently t2 fires: O2 has received a message from O1 After that O1 and O2 execute in parallel independently Compared to other cases, the Petri net for this case is much simple.



(Fig. 3 2) The asynchronous message passing without returning information

The meaning of asynchronous message passing which requires to receive back an information or acknowledgement from the receiver object is given in Figure 3.3. The execution pattern of this statement is very similar to that of synchronous message passing expect that the sender object O1 does not need to be suspended until it receives back a message. However, O1 eventually receives back the information from O2, that is, any of transition (t3, t5, t7, ..) can fire at some time but it does not affect on the execution of the execution of the object O1.



(Fig. 3.3) The asynchronous message passing with returning information

4. Conclusion

One of important concern during the specification and design phase of distributed object-oriented system is how to specify their communication patterns appropriately. Two specification methods for messages passing have introduced. One is the temporal-logic specification method which can specify the different pattern of communication by introducing a set of rules. This methods enables to define the underlying mechanism which can be interpreted as constraints. A set of primitive predicates is pre-defined for this methods and temporal logic like formulae, rules, for each messages passing method are derived The Petri net methods helps to specify graphically the dynamic behavior of communication patterns using the properties of Petri nets. Use of Petri nets in software development is increased.

In the future, we will extend these specification methods to be applied to the entire system specification of distributed object-oriented systems. Furthermore, since there seem to exist similar properties in temporal logic expression and Petri nets, the relationship between temporal logic expression and Petri nets will be investigated

References

[1] H E Bal, "Programing Languages for Distributed Computing System." ACM Computing Surveys, Vol 21, No.3, Sept, 1989, pp 261-322

[2] A. Yonezawa and M. Tokoro, (ed.) 'Object-Oriented Concurrent Programming'. The MIT press. Cambridge, MA, 1987.

[3] G. Booch, "Object-Oriented Development," IEEE Trans. on Soft Eng., SE-12,2 Feb. 1986, pp 211-221.

[4] G. Booch, 'Software Engineering with ADA' (2nd eds.), The Benjamin/Cummings, Redwood city, CA, 1991.

[5] C.A.R Hoare, 'Communicating Sequential Processes', Prentice-Hall Int, 1985.

[6] A Yonezawa (ed), 'ABCL An Object-Oriented Concurrent System', The MIT Press, Cambridge, MA, 1990.

[7] A Pnueli, "The Temporal Logic of Concurrent Programs," Theoretical Computer Science, Vol.13, 1981, pp.45-60

[8] S Lee and D.L. Carver, "Specification of Distributed Systems with Object-Based Specification Language. DOSL," the Technical Report, Louisiana State University, July 1992.

[9] S Lee, 'A Formal Methodology for the Specification of Distributed Systems from an Object Perspective'. Ph D Dissertation, Louisiana State University, 1992.

[10] B Banieqbal, H Barringer and A. Pnueli (eds.), "Temporal Logic in Specification," LNCS, Vol.398, Springer Verlag, 1989.

[11] H Barringer, "Using Temporal Logic in the Compositional Specification of Concurrent Systems," University of Manchester, TR UMCS-96-10-1, 1986.

[12] A. Corradi, and L Leonard, "Parallelism in

Object-Oriented Programming Languages," IEEE 1990 Int'l Conference on Computer Languages, 1990.

[13] C. Petri, *'Kommunikationmit Automen'*, *Ph.D* Dissertation, University of Bonn, Bonn, West Germany, 1962.

[14] S.J. Song, 'The Modeling, Analysis, and Design of Distributed Systems Based on Communication', Department of Electrical Engineering and Computer Sciences, UC Berkeley, 1988.

[15] J.L. Peterson, 'Petri Net Theory and the Modeling of Systems', Prentice-Hall, Inc., Englewood Cliffs, NJ. 1981.

이 상 범

e-mail : sblee@anseo.dankook.ac.kr

1983년 한양대학교 기계공학과 졸업(학사)

1989년 미국 Louisiana State University 전산학과(석사)

1992년 미국 Louisiana State University 전산학과(박사)

1983년～1986년 제일모직(주) 프로그래머

1992년～1993년 한국전자통신원, 선임연구원

1993년～현재 단국대학교 전자계산학과 조교수

관심분야 : 객체지향 모델링, 프로그래밍 언어, 영상처리, 분산객체지향시스템, 데이타베이스