Petri net-based analysis on object assignment in distributed object-oriented systems

Wen-Tsung Chang a,1, Chien-Chao Tseng b,*, Wen-Kuang Chou c,2

a Multimedia Laboratory, Special Systems Division, Institute for Information Industry, Taipei, Taiwan, ROC
b Department of Computer Science and Information Engineering, Institute of Computer Science and Information Engineering, National Chiao-Tung University, 1001 Ta Hsueh Road Hsinchu 30050 Taiwan, ROC
c Department of Computer Science and Information Management, Providence University, Shalu, Taiwan, ROC

Received 19 January 1996; revised 13 December 1996; accepted 19 May 1997

Abstract

Object-oriented programming [9], which treats objects as processes in execution, has shown significant effectiveness in distributed systems. This effectiveness is greatly influenced by how objects are assigned to nodes. In this paper, we present a colored generalized stochastic Petri net (CGSPN) model to analyze the behavior of object invocations when an assignment strategy is applied. The effectiveness of an object assignment is also analyzed by our CGSPN model. Moreover, this analysis provides guidelines to develop an efficient object assignment strategy. [4–8]

Keywords: Petri net; Distributed Systems; Assignment strategy; Object-oriented programming

1. Introduction

Distributed object-oriented systems are composed of number of heterogeneous or homogeneous processing nodes that are linked to an interconnection network (see Fig. 1). Objects in nodes cooperate to accomplish a given task, and objects in different nodes interact with each other via invocations [1,2]. However,

* Corresponding author. E-mail: cctseng@csie.nctu.edu.tw
1 E-mail: wtchang@iii.org.tw
2 E-mail: wkchou@simon.pu.edu.tw

© 1998 Published by Elsevier Science B.V. All rights reserved.
 pii: S1383-7621(97)00049-0
the invocation overhead between nodes is a major bottleneck that affects overall performance. To minimize such overhead, we should first analyze the behavior of objects handled in a distributed manner.

There are two approaches to modeling the behavior of distributed object-oriented systems: the queueing networks (QNs) theory and the generalized stochastic Petri net (GSPN) model. The QNs [10–13] approach is to model objects as servers with queues processing incoming requests, whereas the GSPN [14–20] approach is to model objects with states, transitions and the notation of stochastic process. The GSPN model gives a better description of how transitions, concurrency, and synchronization behave in distributed systems. However, most of them were designed to model the internal behavior of objects in specific languages [17–20]. In our study, we intend to analyze the communication overhead of a distributed object-oriented system. Therefore, we focus on the behavior of interaction among objects rather than the internal behavior of objects. We develop a generalized modeling technique based on the colored GSPN (CGSPN) model since the CGSPN model can clearly describe the behavior of distributed object-oriented systems [21,22]. Moreover, we further use our model to analyze the factors to the effectiveness of an assignment strategy in a distributed object-oriented system.

The rest of this paper is organized as follows: Section 2 introduces our CGSPN model. Section 3 further describes our CGSPN model in a semantic structure. Section 4 verifies our CGSPN model and discusses the effectiveness evaluation of an assignment strategy based on our CGSPN model, and Section 5 presents our conclusions.

2. The CGSPN modeling of object invocations

2.1. An abstract object model

Before describing our CGSPN model, we should first define an abstract object model. From the viewpoint of programming languages, Snyder defined an abstract model based on the following concepts [23]:
- An object explicitly embodies an abstraction (class) characterized by services or operations (methods).
- Operations can be generic; an operation can be uniformly performed with visibly different behaviors on a range of objects (polymorphism).
- Objects can be classified by their services, forming a class hierarchy.
Objects can share the same implementation, either in full (class instances) or in part (class inheritance). To analyze the execution behavior of objects, we further define an object as follows:

- Objects are units of execution, with independent storage containing local variables and associated operations (methods) that maintain these variables.
- Each object belongs to a class, i.e., an object is an instance of a certain class.
- An object is activated by incoming invocations. If the required method is not found in its local storage, the object will bypass this invocation up to its superclass, until the required method is found, or a failure message is returned.

To simplify the analysis of our abstract object model, we make some assumptions about the behavior of object invocations:

1. An object can only execute one invocation at a time, that is, an object has a queue collecting all types of method invocations. Invocations are executed in FCFS (First-Come-First-Served) order, without pre-emption or priority.
2. To ensure consistency in execution, data access in an object is a critical section managed by an operating system, and programs in this operating system are assumed to be deadlock-free.
3. The arrivals of invocations are Poisson processes.

Using the above assumptions, we have proposed a five-phase invocation protocol to describe the interaction behavior of objects handled in a distributed manner [3]:

- **Phase 1**: Start invocation (Issue).
- **Phase 2**: Route invocation to target object (Transmit).
- **Phase 3**: Carry out the appropriate computations (Execute).
- **Phase 4**: Branch to nested invocations and continue execution (Branch).
- **Phase 5**: Return (Return).

This five-phase protocol is developed based on the four-phase protocol which indicates the operation of object invocations by Tomlinson et al. [24]. When a source object activates an invocation to a target object in phase 1, this invocation travels through nodes in phase 2 if the source and target objects are not located in the same node. In phase 3, the target object performs the operations specified in the associated method. If such invocation activates further invocations, the protocol enters phase 4, which recursively repeats phases 1–5, until further invocations have been completed; the target object returns the results in phase 5 after the execution is finished.

Because of its generality, this five-phase protocol can be applied to both the statically typed programming languages, like Eiffel and C++, and the dynamically typed programming languages, like Smalltalk-80 and Common Lisp Object System (CLOS). Moreover, this description can also be applied to develop our analytical model.

2.2. The CGSPN invocation model

As mentioned earlier, CGSPN can be applied effectively to model distributed object-oriented systems since it clearly describes the dynamic behavior of invocations with different colors of tokens. In this section, we propose a model based on CGSPN to analyze the dynamic behavior of object invocations.

In a distributed system, objects are assigned to nodes to perform certain tasks in parallel by an assignment strategy. An assignment strategy can be represented with a mapping function. The mapping function
Map is depicted as \( \text{Map: } \text{Obj} \rightarrow \text{Node} \), mapping function of an assignment strategy, where \( \text{Obj} \) is the set of objects, and \( \text{Node} \) is the set of nodes.

This function maps an object to a certain node. Our CGSPN model is based on the description of nodes since we want to describe the behavior of objects among nodes. The CGSPN model for a particular node \( \mathcal{N}_i \) in a distributed system, denoted as \( \mathcal{ND}_i \), is defined as follows:

**Definition 1.** CGSPN \( \mathcal{ND}_i = (\pi_i, \mathcal{T}_i, \mathcal{A}_i, \mathcal{L}_i, \mathcal{X}, \mathcal{M}_i, \mathcal{M}_0^i) \), where \( \pi_i = \{P_{1i}, P_{2i}, P_{3i}\} \) is the set of places (states of invocation behavior), \( \mathcal{T}_i = \{t_{1i}, t_{2i}, \ldots, t_{r_{1i},0}, t_{r_{2i},0}, \ldots, t_{r_{n_i}-1,0}, t_{r_{n_i},0}, \ldots, t_{r_{m_i},n_1-1}\} \) the set of transitions \( (n = \text{number of nodes in target system}) \), \( \mathcal{A}_i \subseteq \{(\mathcal{P}_i \times \mathcal{T}_i)\} \cup \{(\mathcal{T}_i \times \mathcal{P}_i)\} \) the set of arcs connecting places and transitions, \( \mathcal{L}_i = \{\mu_i, \mu_i^n\} \) the set of method \( m \) firing rates associated with timed transitions, \( \mathcal{X} = \{x_1, \ldots, x_k\} \) the set of token colors \( (k = \text{number of methods in the class hierarchy}) \), \( \mathcal{A}_i: \pi_i \rightarrow \mathcal{X}^* \) the function indicating the numbers and colors of tokens in a given place, and \( \mathcal{M}_0^i \) is the initial marking of a node \( \mathcal{N}_i \).

Tokens in different places stand for states of invocation behavior according to our protocol. A transition is enabled when a sufficient number of tokens are accumulated in all its input places. When a transition is enabled, it may fire immediately, or after a period of time. The duration of time period is determined by the set of firing rates \( \mathcal{L}_i \). Firing a transition may change the color of a token by firing rules. We assume that the firing rules are determined by a source program, and whenever an object method is invoked, its codes can be found in its local processing node.

The CGSPN model \( \mathcal{ND}_i \) only represents the description of a node. In general, an distributed object-oriented program consists of several nodes. Moreover, we need constructs to control invocation activation and variable access. Hence the CGSPN model of an object-oriented program, denoted as \( \mathcal{DS} \), can be depicted as follows:

**Definition 2.** CGSPN \( \mathcal{DS} = (\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{L}, \mathcal{X}, \Lambda, \mathcal{M}_0) \), where \( \mathcal{P} = \cup_{i=0}^{n-1} \mathcal{P}_i \cup \{P_0, \text{ARM}, \text{VM}\} \), \( \mathcal{T} = \bigcup_{i=0}^{n-1} (\mathcal{T}_i \cup \{b_{ri}\} \cup \{t_{fi}\}) \), \( \mathcal{A} \subseteq \{(\mathcal{P} \times \mathcal{T})\} \cup \{(\mathcal{T} \times \mathcal{P})\} \), \( \mathcal{L} = \bigcup_{i=0}^{n-1} \mathcal{L}_i \), \( \Lambda: \left\{ \begin{array}{ll} (\mathcal{P}^{n-1}) \cup \{P_0\} & \rightarrow \mathcal{X}^* \\ \text{ARM} & \rightarrow \{c_a\}^* \\ \text{VM} & \rightarrow \{c_o\}^* \\ M_0 & = \{\Lambda(P_0), \Lambda(\text{ARM}), \Lambda(\text{VM}), M_0^0, \ldots, M_0^{n-1}\} \end{array} \right. \),

where ARM is a place to store activation records of invocations, VM a place to store object variables, \( c_a \) a token of an activation record, and \( c_o \) is a token of variables in an object.

The detailed description of the CGSPN model \( \mathcal{DS} \) is shown in Fig. 2. Function \( f \) is a probability function defined by source program to enable/disable the firing of a transition. This model consists of the descriptions of classes (we use the term “subnet \( \mathcal{ND}_i \)” as the CGSPN model of a node \( \mathcal{N}_i \)). The \( \mathcal{DS} \) model also includes additional places VM and ARM. Tokens in place VM are used as the synchronization mechanisms for variable accesses. A token with color \( c_o \) in VM represents the variables of an object. Tokens with color in place ARM represent the activation records of invocations. These records are used to hold the context of parent invocations. Hence \( \mathcal{X}' \) includes \( X, c_o \), and \( c_a \). Moreover, tokens in VM and ARM are managed by the operating system. In an object-oriented program, objects are activated by invocations. Every invocation
Fig. 2. CGSPN description of the activity of a node \( N_i \) and the system resources.

represents a token in the DS model, and is initially placed in \( P_0 \). Thus, the initial marking \( A(P_0) \) is determined by the initial object assignment in a distributed system. When the execution starts, these tokens enter the places \( P_1 \)'s of associated subnets through transitions \( t_{s1} \)'s according to the codes of the program.

Without loss of generality, assume that a token \( t_{k0} \) is in the place \( P_1_i \) of subnet \( ND_i \) and it invokes an invocation of method \( m \) to object \( O_{ji} \). This token \( t_{k0} \), along with a variable token released from VM, causes the firing of transition \( t_{l1} \) after a time duration (phase 1: Issue). This duration is assumed to be an exponential distribution with parameter \( \mu_i^m \) which depends on the type of method \( m \) and the target class \( i \). Moreover, a new token \( t_{k1} \), which replaces \( t_{k0} \), enters place \( P_2_i \).

At this moment, token \( t_{k1} \) enables all transitions \( t_{m1,i,j} \)'s, as shown in Fig. 2. However, as \( t_{k1} \) is transmitted to the target place \( P_1_j \) of subnet (phase 2: Transmit), only one transition will fire. The firing of transition is determined by function \( f \), where \( f: T \rightarrow \{0, 1\} \). The duration time for transmission is also an exponential distribution with parameter \( \lambda \).

However, \( t_{k1} \) has to wait in \( P_1_j \) of subnet \( ND_j \) if no token of object variables is released from VM (i.e., the target object is accessed by other object, which causes mutual exclusion of critical section in object variables). When a token in VM is released, both transitions \( t_{l1} \) and \( t_{2} \) can be enabled. At this time, firing function \( f \) determines whether token \( t_{k1} \) completes execution or further activates non-local invocations. For the former case, that is, \( t_{k1} \) completes execution, \( f(t_{2,i}) \) will become one and transition \( t_{2,i} \) will fire accompanied with an associated activation record retrieved from ARM. After a duration time of execution (phase 3: Execute), \( t_{k1} \) enters place \( P_3_j \). At this time, function \( f \) causes transition \( tr_{j,i} \) to fire and \( t_{k1} \) traverses back to its parent subnet \( ND_i \) (phase 5: Return).
For the latter case, that is, $tk_i$ activates further non-local invocations, $t1_i$ will fire (phase 3: Execute) and $tk_i$ is stored in ARM as a token of activation record. A new token, namely $tk_j$, indicating further non-local invocation, replaces $tk_i$ (phase 4: Branch) and repeats the process described above. After further non-local invocation completes, token $tk_j$, retrieved from ARM, enters place $P1_j$ of subnet $ND_j$ to continue the remaining process. The process of Branch phase repeats until all the non-local invocations complete. At this time, Return phase starts and token $tk_i$ goes back to parent subnet $ND_i$ through $P3_j$ and transition $tr_{ij}$.

The proposed DS model can be constructed from the codes of an object-oriented program and the associated class hierarchy. In Section 3 we present the semantic constructs of our GSPN model to assist the performance analysis.

3. Semantic constructs of CGSPN model DS

As mentioned earlier, we have proposed a CGSPN model to describe the behavior of object invocations. In our proposed model, every token of invocation has an attribute to show its own behavior. This attribute can be represented by an attribute function, denoted as $I_{attr}$. This function is defined as follows:

$$I_{attr} : Token \rightarrow (Obj \times Obj \times Method \times (Token \cup \{null\})),$$

where $Token$ is the set of tokens in places $P_o, P_{1i}', P_{2i}'s$ and $P_{3i}'s$ ($i = 0, \ldots , n - 1$), $null$ the empty set in places $P_o, P_{1i}', P_{2i}'s$ and $P_{3i}'s$, $Obj$ the set of objects, and $Method$ is the types of method invocations.

The values of these functions are determined by the source program. Moreover, the last term $Token$ in function $I_{attr}$ indicates the token of parent invocation.

If an object $O_{il}$ is assigned to node $N_i$, the mapping function can be defined as $Map(O_{il}) = N_i$. Moreover, a token $tk_0$ in $P_0$ has an attribute which indicates the initial status of program execution, such as $I_{attr}(tk_0) = (O_{il}, m, null)$, where $tk_0$ is an invocation of method $m$ to object $O_{il}$. When the program starts execution, $tk_0$ in $P_0$ moves to $P_{1i}$ of subnet $ND_i$, through transition $ts_i$. After $tk_0$ enters place $P_{1i}$, transitions $t_{1i}$ and $t_{2i}$ are enabled. The firing of $t_{1i}$ and $t_{2i}$ is determined by the firing function $f$. This function decides whether such invocation returns back to parent subnet or further activates inter-node invocations, that is $f(t_{1i}) + f(t_{2i}) = 1$, for all $i, i = 0, \ldots , n - 1$. If $f(t_{2i})$ is one, that is, the invocation completes execution, $t_{2i}$ will fire by retrieving an associated token of activation record from ARM and the resulting token will enter $P3_i$ (see Fig. 2). However, if $f(t_{1i})$ is one, $t_{1i}$ will fire since and $tk_0$ is stored in ARM as a token of activation record, which will be discussed later. At the same time, a new token $tk_i$ is created in $P2_i$ to represent further invocation.

In general, the semantic of transition $t_{1i}(i = 0, \ldots , n - 1)$ is as follows:

For a token $tk$ selected from $P_{1i}$ of subnet $ND_i$, and $I_{attr}(tk) = (O_{il}, O_{il}', m, ptk)$, where $Map(O_{il}) = N_i$

$$f(t_{1i}) = 1 \quad tk \rightarrow tk', \quad \text{and} \quad I_{attr}(tk') = (O_{il}, O_{il}', m', ptk'),$$

where

$$tk, tk' \in Token, \quad ptk \in Token \cup \{null\}.$$  

Whenever a token of an invocation moves from $P_{1i}$ to $P_{2i}$ through $t_{1i}$, its attribute determines the value of firing function. This firing function can be defined as follows:

Let $tk$ be the token in $P_{2i}$ of subnet $ND_i$, $I_{attr}(tk) = (O_{il}, O_{il}', m, ptk)$, $N_i = Map(O_{il})$, and $ptk \in Token \cup \{null\}$,
\[
f(t_{m,i,j}) = 1 \quad \text{if } N_j = \text{Map}(O_{i,j}),
\]
\[
f(t_{m,i,j}) = 0 \quad \text{otherwise},
\]
\[
\text{where } \sum_{i=0}^{n-1} f(t_{m,i,j}) = 1, \quad \text{for all } i, i = 0, \ldots, n - 1.
\]

If \( f(t_{2,i}) \) is one, the firing function \( f \) also determines the return path via transitions \( t_{r,j} \) by the attribute of \( t_{k_0} \), which will be discussed later.

As mentioned in our five-phase protocol, two phases need to access tokens in VM: Issue and Execute. The access control occurs in the transitions \( t_{1,i} \)'s and \( t_{2,i} \)'s. To fire these transitions, the corresponding variable tokens should be found in VM. Each object is associated with a token in VM. The attribute function \( V_{\text{attr}} \) for the tokens in VM is
\[
V_{\text{attr}} : \text{Token}_1 \to \text{Obj},
\]
where \( \text{Token}_1 \) is set of tokens in place VM.

Each object \( O_i \) thus has a token \( v_i \) in VM with attribute \( V_{\text{attr}}(v_i) = O_i \). The conditions required to enable transitions \( t_{1,i} \)'s and \( t_{2,i} \)'s can be described as
Let \( t_k \) be a token of invocation, and \( I_{\text{attr}}(t_k) = \langle O_{i,j}, O_{j,k}, m, ptk \rangle \).

For transitions \( t_{1,i} \) and \( t_{2,i} \):
\[
[t_k \in \Lambda(P_{1,i}) \text{ and } [O_{j,k} \in S_{\text{VM}}],
\]
where \( S_{\text{VM}} = \{O_i \mid \forall v_i \in \text{VM}, V_{\text{attr}}(v_i) = O_i\}, tk \in \text{Token}, ptk \in \text{Token} \cup \{\text{null}\}, \)
\[
v_i \in \text{Token}_1.
\]

When the condition is satisfied, the associated transition retrieves the variable token from VM and, after it has fired, the transition releases this token back to VM.

A token in ARM represents the activation record of an invocation. ARM could be a stack or hash table. Activation records can be retrieved by the function \( \text{Acc} \), denoted as
\[
\text{Acc} : \text{Token} \to \text{Token} \cup \{\text{null}\}.
\]
Initially, we assume that \( \text{Acc}(t_k) = \text{null} \) for any token \( t_k \) in \( P_0 \). Whenever the transition \( t_{1,i} \) fires, token \( t_k \) of method \( m \) in \( P_{1,i} \) will be stored in ARM and create a token of child method \( m' \) invocation \( t_{k'} \) in \( P_{2,i} \).

Hence the semantic of transition \( t_{1,i} \) can be depicted as \( \text{Acc}(t_{k'}) = t_k \), where
\[
I_{\text{attr}}(t_k) = \langle O_{i,j}, O_{j,k}, m, ptk \rangle \quad \text{and} \quad I_{\text{attr}}(t_{k'}) = \langle O_{j,k}, O_{k,l}, m', tk \rangle.
\]

As stated previously, if \( f(t_{2,i}) \) is one, transition \( t_{2,i} \) will try to retrieve a token from ARM via function \( \text{Acc} \). The resulting token then enters \( P_{3,i} \) and the function \( f \) determines the designated transition which the resulting token will traverse, either back to \( P_0 \) or its upper-level subnet. Function \( f \) is defined as
Let \( t_{k'} \) be a token in \( P_{3,i} \) of subnet \( N_{D_i} \), \( I_{\text{attr}}(t_{k'}) = \langle O_{j,k}, O_{k,l}, m', tk, N_k \rangle = \text{Map}(O_{k,l}) \) and \( N_j = \text{Map}(O_{j,k}) \),
\[
f(t_{r_{x,j}}) = 1 \quad \text{if } \text{Acc}(t_{k'}) = t_k \text{ and } I_{\text{attr}}(t_k) = \langle O_{i,j}, O_{j,k}, m, ptk \rangle,
\]
\[
f(t_{r_{x,j}}) = 0 \quad \text{otherwise}.
\]
\[
f(t_{f,i}) = 1 \quad \text{if } \text{Acc}(t_k) = \text{null}.
\]
\[ f(t_f) = 0 \quad \text{otherwise}, \]

where \( \sum_{i=0}^{n-1} f(t_{f,i}) + f(t_f) = 1 \) for all \( x \) and \( j, x, j \), \( \leq 0, \ldots, n - 1 \). (8)

After the designated transition has been determined, the resulting token \( t_k' \) is replaced by the token \( t_k \) in transition \( t_2_i \) if \( \text{Acc}(t_k') = t_k \). Otherwise, \( t_k' \) remains unchanged if \( \text{Acc}(t_k') \) is null.

With the above constructs, we can formally describe the behavior of an object invocation. In Section 4, we will prove that our semantic constructs are correct in the DS model and analyze the effectiveness of an assignment strategy by our DS model.

4. Discussions about the CGSPN model DS

4.1. Correctness of the semantic constructs

In Section 2 we have shown the five-phase protocol of an invocation. This five-phase protocol can also be viewed as a syntax term \( \text{Invoc}(i,j,m) \), defined as follows:

\[ \text{Invoc}(i,j,m) ::= \text{Issue}(i,j,m) \text{Transmit}(i,j,m) \text{Execute}(i,j,m) \]
\[ \quad \text{Branch}(i,j,m) \text{Return}(i,j,m), \]
\[ \text{Execute}(i,j,m) ::= \text{Lookup}(i,j,m) \text{M-Execute}(i,j,m) \]
\[ \text{Branch}(i,j,m) ::= \emptyset | \text{Invoc}(j,k,m') \text{R}(i,j,m), \]
\[ \text{R}(i,j,m) ::= \emptyset | \text{M-Execute}(i,j,m) \text{Branch}(i,j,m) | \text{M-Execute}(i,j,m) \]
\[ \quad \text{Branch}(i,j,m), \]

where \( i, j, k \) are source and target object indices, \( m, m' \) types of methods, \( \text{Issue}( ) \) an atomic operation for phase \( \text{Issue} \), \( \text{Transmit}( ) \) an atomic operation for phase \( \text{Transmit} \), \( \text{Execute}( ) \) a composite operation for phase \( \text{Execute} \), \( \text{Lookup}( ) \) an atomic operation for method lookup in phase \( \text{Execute} \), \( \text{M-Execute}( ) \) an atomic operation for code execution in phase \( \text{Execute} \), \( \text{Branch}( ) \) a composite operation for phase \( \text{Branch} \), and \( \text{Return}( ) \) an atomic operation for phase \( \text{Return} \).

Using this syntax, we deduce three lemmas to prove the correctness of our constructs.

Lemma 1. Semantics of variable accesses is correct for all types of invocations.

The accesses of tokens in VM occur at \( \text{Execution} \) phase since the variables of target object could be collected and updated. The accesses are caused by the firing of transitions \( t_{1,i}' \)s or \( t_{2,i}' \)s. Suppose an invocation of method \( m \) to object \( O_{j_i} \) of node \( N_j \) activates, a token \( t_{k_0} \) enters place \( P_{1_j} \) of subnet \( N_{D_j} \). According to our semantic constructs, condition (5) states that if \( t_{k_0} \) is in place \( P_{1_j} \) and variable token of \( O_{j_i} \) is contained in VM, both \( r_{1_j} \) and \( t_{2_j} \) are enabled and one of them is fired by function \( f \). Thus, we can see that the semantic constructs of VM are correct.

In Lemma 2 we prove that the semantic constructs of accessing ARM are also correct. Since the accesses of tokens in place ARM occur only at transitions \( t_{1,i}' \)s and \( t_{2,i}' \)s, it is thus sufficient to show that the semantics is correct in transitions \( t_{1,i}' \)s and \( t_{2,i}' \)s.
Lemma 2. Semantics of accessing ARM is correct for all types of invocations.

Invocations can be classified into two types: one is initial invocations contained in the main program, and the other is activated by other invocations. For the first type of invocations, the initial values of Acc( ) are null, while the values for the second type are not. Suppose that an invocation of method m to object Oj of node Nj, denoted as Invoc(ij, j1, m), is activated and a token tkj enters place P1j of subnet NDj. At this time, both t1j and t2j are enabled if condition (5) is satisfied. The behavior of tkj depends on the following values of f( ):

Case 1 (f(t2j) = 1): t2j fires by retrieving a token from ARM. If tkj is an initial invocation, Acc(tkj) is null and function f causes the firing of transition t fj by statement (8), that is

\[ f(tfj) = 1 \quad \therefore \text{Acc}(tkj) = \text{null}, \quad N_j = \text{Map}(O_{ji}). \]

Token tkj will go back to P0 through transition t fj and terminates its execution.

If tkj is the second type of invocations, Acc(tkj) is not null, and suppose that it is tk0. Function f causes the firing of transition trj,i by statement (8), that is

\[ f(trj,i) = 1 \quad \therefore \text{Acc}(tkj) = tk0, \quad I_{attr}(tr0) = (O_{i1}, O_{j1}, m, ptk), \quad \text{Map}(O_{i1}) = N_i. \]

Token tk0 will replace tkj and go back to P1j of subnet NDj through transition trj,i to continue the remaining process of tk0.

Case 2 (f(tlj) = 1): tlj fires to further activate a non-local invocations. Suppose in node Nj, tkj further invokes method m' to object Oi1 of node Ni, denoted as Invoc(j1, k1, m'), where Nj \( \neq \) Ni. At this time, tkj is stored in ARM and a token tkj of Invoc(j1, k1, m') is created, and Acc(tkj) is set as tkj. After the firing of transition tlj, token tkj enters place P1j of subnet NDj through t mj,i.

When Invoc(j1, k1, m') completes, tkj enters P3j of subnet NDj through transition t2j (since f(t2j) is one). By statement (8) function f causes the firing of transition trj,i since Acc(tkj) is tkj, not null. Therefore, tkj replaces tkj, and correctly goes back to P1j of subnet NDj.

After Invoc(i1, j1, m) completes Branch phase, f(t2j) becomes one and repeats the process of case 1. Thus these semantic constructs are correct for all types of invocations.

Since the Branch phase includes a composite operation R( ) in syntax, we need to prove that R( ) works correctly for all types of invocations.

Lemma 3. The proposed semantics is correct in R( ).

By the definition of invocations, R( ) represents the remaining process of an arbitrary invocation. There are four cases for the derivation of R( ): \( \emptyset \), M-Execution( ), Branch( ) and M-Execution( ) Branch( ). Since the time duration of Execution( ) can be zero, \( \emptyset \) is thus a special case of M-Execution( ), and Branch( ) is also a special case of M-Execution( ) Branch( ). For a subnet NDj, the first two cases happen when f(t2j) is one and a token of an invocation will directly enter P3j by firing transition t2j. At this time, R( ) completes its process and our semantic constructs are thus correct.
The latter two cases happen when \( f(t_{1i}) \) is one, \( t_{1i} \) will fire and activate further invocation. Since the firing of \( t_{1i} \) indicates the process of a further \( \text{Branch}() \), which causes another process of \( R() \), our constructs are thus also correct.

With the preceding lemmas, we can prove the correctness of our semantics with the following theorem.

**Theorem 1.** The proposed semantic constructs are correct for all types of invocations.

**Proof.** Without loss of generality, let us examine the behavior of an invocation which activates arbitrary levels of cascading non-local invocations, as shown in Fig. 3. Let us examine an \( a \)th level invocation, denoted as \( \text{Invoc}(i_{a-1}, i_a, m_a) \), which is represented by a token \( t_{k_a} \) in our DS model. This token indicates a non-local invocation of method \( m_a \) from object \( O_{i_{a-1}} \) to object \( O_{i_a} \), where \( \text{Map}(O_{i_a}) = N_{x_a}, \ \text{Map}(O_{i_{a-1}}) = N_{x_{a-1}} \), and \( N_{x_a} \neq N_{x_{a-1}} \).

We prove this theorem by induction on \( a \).

(i) **Case \((a = 1)\):** We discuss the behavior of \( \text{Invoc}(i_0, i_1, m_1) \) with the five-phase protocol.
   (a) **Issue, Transmit and Execution:** By Lemmas 1 and 2, since the access of VM and ARM works correctly, the process of these phases is correct.
   (b) **Branch:** After the process of Execution phase, token \( t_{k_1} \) enters \( P_{1x_1} \) of subnet ND_{x1}. If \( t_{k_1} \) does not activate further invocation, the Branch phase will be skipped and Return phase starts directly. If \( t_{k_1} \) does, \( f(t_{1x_1}) \) becomes one and \( t_{1x_1} \) fires. At this time, \( t_{k_1} \) is stored in ARM and token \( t_{k_2} \), which represents the invocation \( \text{Invoc}(i_1, i_2, m_2) \), enters \( P_{2x_1} \) of subnet ND_{x1} (by Lemma 2). \( \text{Acc}(t_{k_2}) \) is then set to be \( t_{k_1} \). After \( \text{Invoc}(i_1, i_2, m_2) \) is completed, token \( t_{k_1} \) replaces \( t_{k_2} \) and enters \( P_{1x_1} \) of subnet ND_{x1} through transition \( tr_{x_2, x_1} \) by statement (8) \( (\therefore \text{Acc}(t_{k_2}) = t_{k_1}) \). Thus, the process of Branch phase is correct.
   (c) **R( ):** By Lemma 3, since there is no further invocation in \( \text{Invoc}(i_0, i_1, m_1) \), we know that our semantics is correct in the process of \( R( ) \) for all types of invocations.
   (d) **Return:** After the execution of Branch, \( t_{2x_1} \) fires and \( t_{k_1} \) enters \( P_{3x_n} \) of subnet ND_{x_n}. By Lemma 2 and statement (8), \( \text{Invoc}(i_0, i_1, m_1) \) thus completes correctly and token of invocation moves back to \( P_0 \) or its parent subnet.

![Fig. 3. Structure of an object invocation.](image-url)
(ii) Case ($a = n$): Assume that our constructs work correctly for $n$-level invocation $\text{Invoc}(i_{n-1}, i_n, m_n)$.

(iii) Case ($a = n + 1$): Suppose that $\text{Invoc}(i_{n-1}, i_n, m_n)$ further activates an invocation $\text{Invoc}(i_n, i_{n+1}, m_{n+1})$.

By case ($a = 1$) $\text{Invoc}(i_n, i_{n+1}, m_{n+1})$ works correctly.

Thus we conclude that our DS model can describe all kinds of invocations in an object-oriented program. □

Theorem 1 can formally prove the correctness of our semantic constructs. In Section 4.2, we formulate a performance model of assignment strategies and derive the guidelines for designing effective object assignment strategies.

### 4.2. Effectiveness of assignment strategies

The effectiveness of an assignment strategy can be measured by the communication and computation costs, which are denoted as $C_{\text{comm}}$ and $C_{\text{comp}}$, respectively. By examining our DS model, it is obvious that $C_{\text{comm}}$ is incurred by the $\text{Transmit}$ and $\text{Return}$ phases (time duration of firing transitions $tm_{i,j}$'s, and $tr_{i,j}$'s), while $C_{\text{comp}}$ by the $\text{Execute}$ phase (time duration of firing transitions $t_1$'s and $t_2$'s). For a distributed object-oriented system, these costs play an important role for the overall system performance. Hence, an effective assignment strategy should minimize these costs.

In this section, we use the analytical measurement to measure the costs based on our DS model. We first assume the following probability values for firing function $f$ as ($m$ indicates type of method):

1. $p\{f(t_{2i}) = 1\} = q$, and $p\{f(t_{1i}) = 1\} = (1 - q)$,
2. $p\{f(m_{i,j}) = 1\} = p^m_{i,j}$, where $\sum_{j=0}^{n-1} p^m_{i,j} = 1$,
3. $p\{f(t_{fi}) = 1\} = r_m$,
4. $p\{f(tr_{i,j}) = 1\} = t^m_{i,j}$, where $\sum_{j=0}^{n-1} t^m_{i,j} + r_m = 1$.

With the definitions of DS model stated in Section 2, we assume that if the code of the associated method cannot be found locally, this invocation will be by-passed to other nodes as a non-local invocation. The probability of this by-passed invocation is defined as $p_{\text{bypass}}$. Thus the mean value of $f(t_{2i})$ becomes $q/(1 + p_{\text{bypass}})$, which is denoted as $q'$. With the above probability values, we can transform our DS model into Markov chains to evaluate $C_{\text{comm}}$ and $C_{\text{comp}}$. The states of these Markov chains are defined as follows:

$$S = (\omega_{a,b})_{n \times 3},$$

where $\omega_{a,b} = (\sigma^1_{a,b}, \sigma^2_{a,b}, \ldots, \sigma^k_{a,b})$ is the set of color tokens in the place of subnet $\text{ND}_a$, $k$ = number of colors and $\sigma^m_{a,b}$ is the number of tokens of color $m$ in the place of subnet $\text{ND}_a$, $\sigma^m_{a,b} \geq 0$, $m$ indicates type of method.

In Fig. 4, we give an example to illustrate the states and the transitions of the Markov chains for an invocation. This example indicates an invocation of a method $m$ from node $N_i$ to node $N_j$ which further activates a method $m'$ to node $N_k$. As shown in Fig. 4, there are seven states, namely $S1$–$S7$. The contents of these seven states are expressed as follows:
Fig. 4. Markov chains for an invocation $m$ issued from node $N_i$ to node $N_j$, $(S_1, \ldots, S_7 \in S)$.

$$S_1 = \begin{bmatrix}
\ldots \\
(\ldots, \sigma^m_{i,1}, \ldots) \\
(\ldots, \sigma^m_{i,2}, \ldots) \\
(\ldots, \sigma^m_{i,3}, \ldots) \\
\ldots
\end{bmatrix},$$

$$S_2 = \begin{bmatrix}
\ldots \\
(\ldots, \sigma^m_{i,1} - 1, \ldots) \\
(\ldots, \sigma^m_{i,2} + 1, \ldots) \\
(\ldots, \sigma^m_{i,3}, \ldots) \\
\ldots
\end{bmatrix},$$

$$S_3 = \begin{bmatrix}
\ldots \\
(\ldots, \sigma^m_{i,1}, \ldots) \\
(\ldots, \sigma^m_{i,2}, \ldots) \\
(\ldots, \sigma^m_{i,3}, \ldots) \\
\ldots
\end{bmatrix},$$
Since we only concern the computation of $C_{\text{comm}}$ and $C_{\text{comp}}$, we eliminate the places $P_{2_i}$'s and $P_{3_i}$'s to simplify the analysis. The set of states $S$ can thus be reduced to a new set of states, namely $S'$, depicted as follows:
The associated simplified Markov chains for the example stated in Fig. 4 are shown in Fig. 5.

With the above descriptions, we can construct the whole simplified Markov chains for a given assignment strategy. Moreover, based on these Markov chains, we can obtain the cost of an invocation of method \( m \) from node \( N_i \) to node \( N_j \), denoted as \( C_{inv}(i, j, m) \), as follows:

\[
C_{inv}(i, j, m) = \frac{2}{\lambda} + 1\mu_i^m + 1\mu_j^m.
\]  

(9)

Besides, in the main program, we can also compute the startup cost of an initial method \( m \) invocation in node \( N_i \), denoted as \( C_{start}(i, m) \)

\[
C_{start}(i, m) = \frac{1}{\mu_i^m}.
\]  

(10)

Moreover, we can further obtain the average total cost of an object-oriented program, namely \( \overline{C}_{prg} \), as

\[
\overline{C}_{prg} = \sum_{i=0}^{\infty} \{p(l_t)l_t \sum_{i,j,m} C_{inv}(i, j, m)\} + \sum_{i=0}^{\infty} \{p(l_s)l_s \sum_{i,m} C_{inv}(i, m)\},
\]  

(11)

where \( p(l_s) \) is the probability of \( l_s \) startup invocations in the main program and \( p(l_t) \) is the probability of \( l_t \) inter-node invocations running in the target system. From the above cost functions, we can observe from the Markov chains that \( C_{comp} \) comes from the duration of \( \mu_i^m \) and \( C_{comm} \) comes from the duration of \( \lambda \). We can also conclude that if the probability \( p_{bypass} \) decreases, that is, the probability of finding required method codes locally increases, \( q' \) also increases, and cost of invocations thus decreases.

In our DS model, the parameters \( p_{ij}^m \)'s, \( t_{ij}^m \)'s and \( \lambda \) can be determined by the number of nodes \( n \) and the topology structure \( G \) in the target system, while the other parameters can be determined by the assignment strategies. Therefore, \( p_{ij}^m \)'s, \( t_{ij}^m \)'s and \( \lambda \) can be depicted as \( p_{ij}^m(n, G) \), \( t_{ij}^m(n, G) \) and \( \lambda(n, G) \). For a static assignment strategy, the parameters \( \mu_i^m \) and \( q' \) are fixed since objects are assigned to nodes before execution.
However, for a dynamic assignment strategy, since objects are created, assigned or destroyed in run-time, $\mu_i^n$ and $q'j$ also vary in run-time. Hence, $\mu_i^n$ and $q'j$ can be viewed as time-varying functions, denoted as $\mu_i^n(t)$ and $q'(t)$.

To reduce $\overline{C_{\text{evg}}}$ caused by these by-passed invocations, there are two approaches: duplicating all the necessary method codes invoked in a node, or grouping objects with sub- or super-class relation in a node. For the first approach, the by-passed invocations can be eliminated, however the total space cost will be increased due to redundant code duplication. For the second approach, the space cost will be minimized, however the cost of inter-node invocations may not be minimized since objects that interact frequently are usually of different classes (without superclass or subclass relation). Therefore we should minimize the combined cost of space and by-passed invocations in designing an effective object assignment strategy.

5. Conclusions

In this paper we propose a model DS to describe the behavior of object invocations when an assignment strategy is applied. This model also depicts the detailed phase transitions for various kinds of object invocations. It should be noted that our model permits multiple invocations runs in parallel to simulate the behavior of a distributed object-oriented system.

There are many ways to measure the costs of a system. In this paper, we applied the analytical measurement by our CGSPN model DS for an object-oriented program in a distributed manner. In our five-phase protocol, we observed that the communication cost results from the Transmit and Return phases (caused by transitions $tm_{i,j}$'s and $tr_{i,j}$'s), whereas the computation cost results from the Execute phase (caused by transitions $r1_{i,j}$'s and $r2_{i,j}$'s). Finally, we provided guidelines to evaluate a given assignment strategy. Such guidelines are helpful in designing an effective object assignment strategy.

References


Wen-Tsung Chang is currently a senior engineer and a project manager in Multimedia Laboratory, Institute of Information Industry, Taipei, Taiwan, ROC. He received his B.S, M.S and Ph.D. degrees in Computer Science from National Chiao-Tung University, in 1989, 1991 and 1995, respectively. His research interests are object-oriented design and programming, performance evaluation and distributed systems. Currently, he is interested in applications of multimedia and computer-aided architectural design.

Chien-Chao Tseng is currently a professor in the Department of Computer Science and Information Engineering at National Chiao-Tung University, Hsin-Chu, Taiwan. He received his B.S. degree in Industrial Engineering from National Tsing-Hua University, Hsin-Chu, Taiwan, in 1981; M.S. and Ph.D. degrees in Computer Science from the Southern Methodist University, Dallas, Texas, USA, in 1986 and 1989, respectively. His research interests are in Mobile Computing, and Parallel and Distributed Processing.

Wen-Kuang Chou is an associate professor of the Department of Computer Science and Information Management at Providence University, Shalu, Taiwan, ROC. He received B.S. from the National Chiao-Tung University in Computer Engineering, M.S. in Computer Science from National Taiwan University, and Ph.D. from the Southern Methodist University in Computer Science and Engineering, in 1983, 1986, and 1991, respectively. From 1984 to 1986, he was a research assistant in Institute of Information Science at Academic Sinica, Taipei, ROC. Also from 1986 to 1987, he was an assistant research fellow in the same institute. From 1989 to 1991, he was an assistant researcher of Pacific International Center of High Technology Research (PICHTR) in Information Technology Division, Honolulu, Hawaii. At the same time, he was also an assistant researcher of Lab. of Intelligent and Parallel Systems (LIPS) at University of Hawaii. His research interests include applications of artificial intelligence and neural networks, theorems of neural networks, speech and image processing and recognition, parallel architectures in discrete transforms, computer architectures and multimedia systems. Currently, he is interested in the network management and mobile computing.