

Grid Computing Simulation and Verification Based on pi Calculus

Tao Hu¹, Shaofan Chen², and Weibo Lin³

¹ Information Management Department, Tourism College, Hainan University, 58 Renmin Ave, Haikou, Hainan, 570228, P.R. China

² Information Science and Technology college, Hainan University

³ Library of Haian University, linweibo@163.com

Abstract. As weakness exists in describing behaviors and mobility in Petri Net theory, we put forward a mode using pi calculus to describe the grid computing, and present its simulation and verification. We briefly introduce the pi calculus at first. Then, it is drawn in detail how to outline single node computing and grid computing by pi calculus: it can be described as a two-player game. The first player moves the first token from the node to a neighboring node along an outgoing edge. On the other hand, the second player can respond only by moving the other token from the node it is on to adjacent node along an outgoing edge. It can be defined the winning mechanism which is related to the relation of the two computing ways: strong bi-simulation, weak simulation or reduction simulation. Many properties (such as deadlock or mutual exclusion violation) may be checked in this fashion. At last an example scenario is presented.

1 Introduction

To provide security modeling, a modeling language - SJAN is described in [1]. SJAN models its security representation by the simple reductions. Once a scenario has been modeled, it can be easily implemented in the real implementation language. SJAN is also suitable for modeling web-services based computation and grid computing. It is developed a logical framework usable for representing, monitoring and enforcing service contracts like SLAs with a combination of adequate logical formalisms in [2]. It may be used in the automation of contract enforcement processes such as the detection of contract violations, authorization control, conflict detection, service billing and reporting. It is not suitable for grid computing or web-service based computation simulation and verification. It is presented a novel grid Node by Node security model in [3]. The numerous grid nodes are partitioned into different autonomy areas and each area is designated a server. SPI calculus is used to verify the security negotiation process. It is a security model which cannot be used

Please use the following format when citing this chapter:

Hu, T., Chen, S., Lin, W., 2006, in International Federation for Information Processing, Volume 205, Research and Practical Issues of Enterprise Information Systems, eds. Tjoa, A.M., Xu, L., Chaudhry, S., (Boston:Springer), pp.133-142.

for grid computing simulation to reflect the relationship between single node computing and grid computing. A calculus is defined for spatial reasoning on a grid structure in [4]. It also presents a logical calculus, investigates the complexity of the satisfiability problem, and proves its NP completeness and specifies additionally a concrete algorithm for solving it. In the paper [5], an abstract model of agent-based service publication and discovery for resource management in grid context is presented, which is the basis for analyzing service publication and discovery. Qi et al. [6] presents a new calculus called Membrane Calculus for grid transactions based on P systems and Petri Nets. The paper provides a general semantic calculus of Grid transactions which has two distinct advantages: dynamic structure and location mobility.

A task may be completed in a computer (node) – we call it single node computing, and it may be divided into several subtasks and finished in many computers (nodes) – grid computing. But what is the relationship of single node computing solution and grid computing solution for a task? If a task can be done by single node, and whether it can be completed by grid computing? Otherwise, if it can be done by grid computing, and whether it can be completed by single node?

In order to solve these problems, a two-player game simulation is introduced: a two-player game on the directed graph who nodes are the processes and whose arrows are given by the transition relation. The two players representing single node and grid computing respectively move alternately. The play is finite, and one player may be in its final position the player whose turn it is cannot move. If a player wins, then it can simulate the other. If losing, it cannot simulate.

The contributes of this paper are: 1) putting forward a new model to support concurrent grid computing or web serviced based computation simulation and verification; 2) using pi calculus to describe the business transaction processes; 3) presenting the algorithms for discussing the relation of the single node computing and grid computing.

This paper has been organized as follows. In Section 2 we introduce the pi calculus notation and its functionality briefly. In Section 3 we describe the idea of modeling web-services based computation and grid computing. Then grid computing simulation discussion and verification solution based on pi calculus will be addressed in Section 4. An example scenario is provided in section 5. In Section 6 we conclude our work and describe our future plan.

2 π -Calculus

The pi calculus provides a framework for describing concurrent systems and reasoning about their behaviors. The entities of the pi calculus are names and processes. Names can be thought of as communication pipelines. Processes, or agents, use names to interact, and pass names to one another by informing them in interactions. Names received by a process can be used and mentioned by it in further interactions. Processes can be defined as follows [7-10]:

$$P ::= 0 \mid \bar{x}y.P \mid x(z).P \mid \tau.P \mid [x = y].P \mid P + Q \mid P \mid Q \mid \nu zP \mid !P$$

Where,

- (1) 0 is inaction; it is a process that can do nothing;
- (2) The output prefix $xy.P$ sends the name y via the name x and evolves to P ;
- (3) The input prefix $x(z).P$ can receive any name via x and continue as P with the received name substituted for the bound variable z .
- (4) The unobservable prefix $\tau.P$ can evolve invisibly to P . As in CCS (A calculus of Communicating Systems), τ can be thought of as expressing an internal action of a process.
- (5) The match prefix can evolve as P if x and y are the same name, and can do nothing otherwise.
- (6) $P + Q$ are those of P together with those of Q . When one of them is chosen, the other is rendered void.
- (7) $P | Q$. The components P and Q can be executed independently and can interact via shared names. For instance, $xyP | x(z)Q$ has this capability: to send y via name x , to receive a name via x , and evolve to $P | Q(y/z)$ invisibly as an effect of an interaction between its components via the shared name x ;
- (8) In the restriction νzP , the scope of the name z is restricted to P . Components of P can use z to interact with one another.
- (9) Finally, the replication $!P$ can be thought of as an infinite composition $P | P | \dots$.

To make process evolution behavior precise, and it is defined the reduction rule: \rightarrow , on processes. The assertion $P \rightarrow P'$ expresses that process P can evolve to process P' as a result of an interaction that is an action within P . The essence of reduction is captured in:

$$(x y.P_1 + M_1) | (x(z).P_2 + M_2) \rightarrow P_1 | P_2\{y/z\}$$

The process P , on the left of the arrow consists of two components. Process $P_1(x y.P_1 + M_1)$ can send y via x , and process $P_2(x(z).P_2 + M_2)$ receive a name via x . It expresses that P has a reduction arising from an interaction between its components (P_1, P_2) via x . y is passed from the first component to the second and it will substitute the placeholder z in P_2 , the two prefixes are consumed. The other two components, expressed by M_1 and M_2 , are rendered void; in summary, P evolves to $P_1 | P_2\{y/z\}$. Table 1 presents the reduction rules.

Table 1. The Reduction Rules

R-INTER	$(x y.P_1 + M_1) (x(z).P_2 + M_2) \rightarrow P_1 P_2\{y/z\}$
R-TAU	$\tau.P + M \rightarrow P$
R-PAR	$\frac{P_1 \rightarrow P_1'}{P_1 P_2 \rightarrow P_1' P_2}$
R-RES	$\frac{P \rightarrow P'}{\nu zP \rightarrow \nu zP'}$

R-STRUCT	$\frac{P_1 \equiv P_2 \rightarrow P_2' P_1'}{P_1 \rightarrow P_1'}$
----------	---

In pi calculus, its operational semantic can be represented as labeled transition system (LTS).

Definition 1 Labeled Transition System (LTS) LTS can be defined as:

$$LTS = (S, T, \{\xrightarrow{t} : t \in T\})$$

S: set of states (processes, agents)

T: set of transition labels (actions), and may use *Act* instead.

$\xrightarrow{t} \subseteq S \times T \times S$: transition relation

The transitions of each composite process (agent) should follow label transition rules such as:

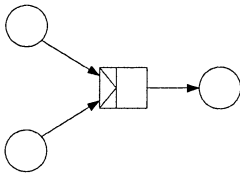
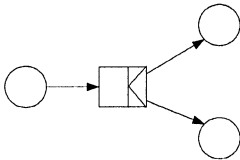
$$\frac{P \xrightarrow{\alpha} P'}{P | Q \xrightarrow{\alpha} P' | Q} \quad bn(\alpha) \cap fn(Q) = \phi$$

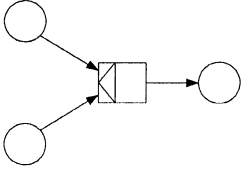
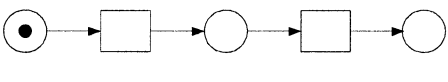
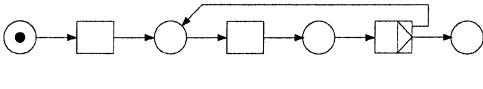
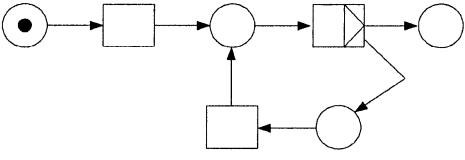
3 Modeling web-services based computation and grid computing

3.1 Expressing workflow basic forms in pi calculus

A task can be described in workflow. There are the following basic workflow forms: AND-join, AND-split, OR-join, OR-split, Iteration and Causality [11]. Table 2 gives their forms in Petri Net and pi calculus.

Table 2. Workflow Basic Forms in pi calculus

Workflow meta	Petri Net	π calculus
And-join		$P = a.R'$ $Q = b.R'$ $P Q \rightarrow R'$ $R' = c.R$
And-split		$P = a.(b.Q c.R)$

<p>Or-join</p>		<p>$P = a.c.R$ $Q = b.c.R$</p>
<p>Or-split</p>	<p>Error! Objects cannot be created from editing field codes.</p>	<p>$P = a.c.Q + b.c.R$</p>
<p>Causality</p>		<p>$P = a.Q$ $Q = b.R$</p>
<p>Do-Iteration</p>		<p>$P = a.Q$ $Q = b.P + b.0$</p>
<p>While-Iteration</p>		<p>$P = a.b.P + a.0$</p>

3.2 Computing Modeling

It can be used to simulate a single node computing and the grid computing between many nodes as a two-player game on the directed graph who nodes are the processes and whose arrows are given by the transition relation. The two players move alternately. The play is finite, and one player may be in its final position the player whose turn it is cannot move. A play begins with two nodes occupied by tokens. The first player moves the first token from the node to a neighboring node along an outgoing edge. On the other hand, the second player can respond only by moving the other token from the node it is on to a neighboring node along an outgoing edge. If the play is infinite, then the second player wins. If after some finite number of steps the player whose turn it is cannot move, then that player loses.

For given starting processes, the second player has a winning strategy for the game if and only if the processes are reduction bi-similar. That is, they are related by a bi-simulation on the graph.

At the beginning of introducing the bi-simulation, we first describe the τ action. Our aim in analyzing the behavior of composite systems is to ignore, as far as possible, their internal actions. We use τ to represent the internal action of a composite process. τ does not represent a potential communication, and is therefore

not directly observable. We regard two processes as equivalent if they exhibit the same (in some sense) pattern of external actions. This amounts to abstracting from such a process just that external aspect of its behavior which is relevant when it occurs as a component of a still complicated process.

The relations of process P (we use it to represent single node computing or grid computing) and Q (the left computing) can be described as Definition 2 –Definition 6.

The transition relations describe how processes can evolve step by step. The transition $P \xrightarrow{\alpha} P'$ expresses that P can evolve to P' by performing the action α , sending or receiving a name. $P \xRightarrow{\alpha} P'$ expresses that P can evolve to P' as a result of an evolution whose content is action α , but which may involve any number of internal actions before or after α .

4 Computing Verification

Pi-calculus can be used to simulate the grid computing, and it can also present grid computing verification. It can use the following algorithms to accomplish this function.

Algorithm 1

- 1) begin.
- 2) divide the task into several sub-tasks according to each node's computing capability;
- 3) present the workflow diagram according to the transaction rules in single node computing;
- 4) present the workflow diagram according to the transaction rules for grid computing;
- 5) change them into the pi-calculus form by LTS (Table 3 - Workflow Basic Forms in pi calculus, Table 2 - Label transition rules and Table 1- reduction rules) obtained in steps 3) and 4), and it can be expressed by P and Q respectively;
- 6) check if existing deadlocks or mutual exclusion violation in them;
- 7) deduce the bi-simulation type of them belong to: weak bi-simulation, reduction bi-simulation or strong bi-simulation defined in section 3. This can be described as a two-player game. P is a player, and Q is the other. If P wins, then Q cannot simulate the grid computing, that is, the grid computing among the nodes are not successful to accomplish the assigned task. If Q wins, the grid computing is succeeding in completing the mission. If there are the above simulations between P and Q, we say Q wins. Otherwise, Q loses;
- 8) end.

5 An Example Scenario

The following is a diagram of describing “open account” for a bank client as shown in Fig 1. Assigning that this job is completed with three nodes (computers), which are client representative (abbr. representative), credit manager (abbr. manager) and client. Here a represents an account.

The grid computing is as following:

$$\begin{aligned}
 \text{Representative} &= \text{in}(a).\text{Start}(a) \\
 \text{Start}(a) &= \text{Collect}(a)
 \end{aligned}$$

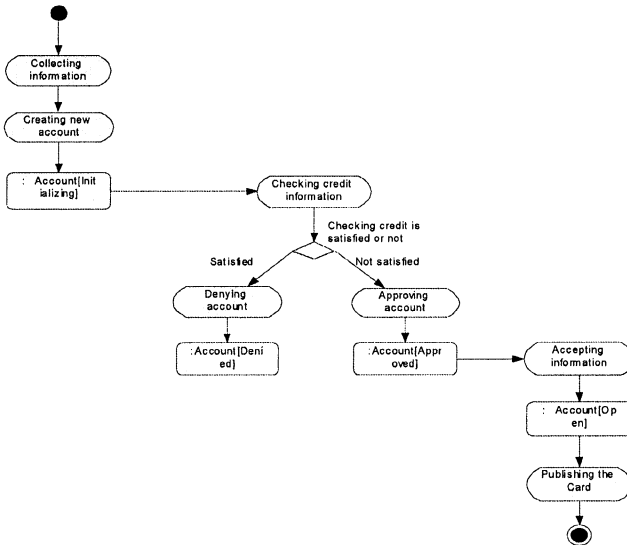


Fig. 1. Activity diagram of Open Account

$$\text{Collect}(a) = \text{Create}(a) \text{ }^{def}$$

$$\text{Create}(a) = \text{Initialize}(a) \text{ }^{def}$$

$$\text{Initialize}(a) = \overline{\text{out}} \text{ (Initializing}(a)\text{).Representative} \text{ }^{def}$$

$$\text{Manager} = \text{in}(b).\text{startEvaluate}(b) \text{ }^{def}$$

$$\text{startEvaluate}(b) = \text{if satisfied}(b) \text{ then Approve}(b) \text{ else Deny}(b) \text{ (Approve}(b) +$$

Deny(b))

$$Deny(b) \stackrel{def}{=} done(b).Manager$$

$$Approve(b) \stackrel{def}{=} \overline{out} (Approved(b)).Manager$$

$$Client \stackrel{def}{=} in(c).startOpen(c)$$

$$StartOpen(c) \stackrel{def}{=} Accept(c)$$

$$Accept(c) \stackrel{def}{=} Open(a)$$

$$Open(c) \stackrel{def}{=} Publish(c)$$

$$Publish(c) \stackrel{def}{=} \overline{out} (done(c)).Client$$

In the other way, this task can be done in a single computer (node). The single node computing is as following:

$$Representative \stackrel{def}{=} in(a).Start(a)$$

$$Start(a) \stackrel{def}{=} Collect(a)$$

$$Collect(a) \stackrel{def}{=} Create(a)$$

$$Create(a) \stackrel{def}{=} Initialize(a)$$

$$Initialize(a) \stackrel{def}{=} \overline{out}$$

$$(Initializing(a)). in(Initializing(a)).startEvaluate(Initializing(a))$$

$$startEvaluate(Initializing(a)) \stackrel{def}{=} \text{if } satisfied(Initializing(a)) \text{ then}$$

$$Approve(Initializing(a)) \text{ else } Deny(Initializing(a)) (Approve(Initializing(a)) +$$

$$Deny(Initializing(a)))$$

$$Deny(Initializing(a)) \stackrel{def}{=} \overline{out} (done(Initializing(a))).Representative$$

$$Approve(Initializing(a)) \stackrel{def}{=} \overline{out} (\quad Approved(Initializing(a)))$$

$$.in(Approved(Initializing(a))).startOpen(Approve(Initializing(a)))$$

$$StartOpen(Approve(Initializing(a))) \stackrel{def}{=} Accept(Approve(Initializing(a)))$$

$$Accept(Approve(Initializing(a))) \stackrel{def}{=} Open(Approve(Initializing(a)))$$

$$Open(a) \stackrel{def}{=} Publish(Approve(Initializing(a)))$$

$Publish(Approve(Initializing(a)))$

def —
 = *out* (*done*(*Approve*(*Initializing*(*a*))).*Representative*

The Mobility Workbench (MWB) is an automated tool for manipulating and analyzing mobile concurrent systems (those with evolving connectivity structures) described in the polyadic pi calculus.

The main feature of this version of the MWB is checking open bi-simulation equivalences, and doing so with high efficiency. The open bi-simulation equivalences are described in a polyadic setting, and efficient characterisations of both the strong and the weak equivalences are illustrated and proven to coincide with their standard formulations.

We use MWB here to check the relations of the above processes as shown in Fig 2. It is revealed that the two processes are strong bi-simulation, and they can simulate each other.

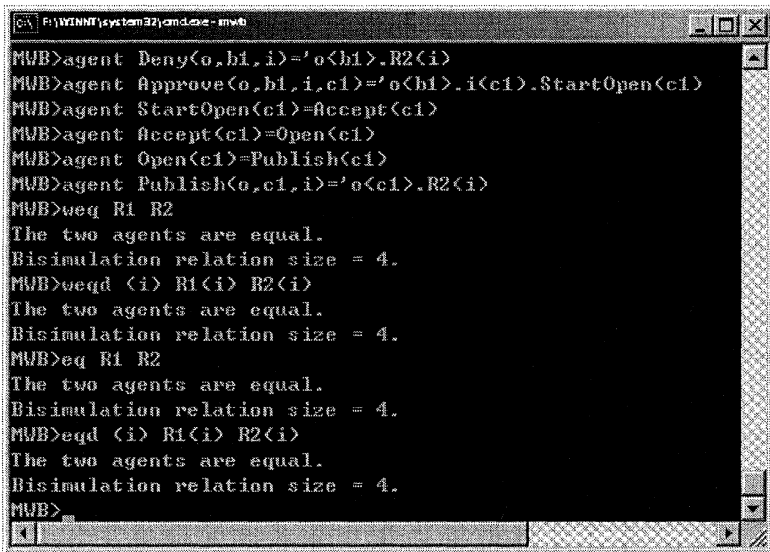


Fig. 2. Computing simulation and verification by MWB

6 Conclusions and Future Work

In order to describe the relationship of single node computing solution and grid computing solution for a task, a two-player game simulation is introduced. At first, a new model is put forward to support concurrent grid computing or web serviced based computation simulation and verification; then pi calculus is used to describe the business transaction processes; finally, it is presented the algorithms for discussing the relation of the single node computing and grid computing. As Petri Net theory lacks of the ability of expressing process behaviors and mobility, this mode supports describing the process behaviors and the communications among the

Processes. Additionally, it also discloses the relationship of single node computing solution of grid computing solution. In the future, we will detail the model checking and verifying functionality to meet the rigorous demands from grid computing.

Acknowledgments

This research is supported by the National Science Fund No. 70561001/G0110 and the Province Science Fund Project “the Research on the collaboration security pattern in e-Government”.

References

1. Y.J. Lee and P. Henderson, A Practical Modelling Notation for Secure Distributed Computation, Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05), pp. 439–442, 2005.
2. A. Paschke and M. Bichler, SLA Representation, Management and Enforcement, Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005.
3. Z.Y. Xia and Y.C. Jiang, A Novel Grid Node-by-Node, GCC 2004, LNCS 3251, pp. 356–363, 2004.
4. M. Ragni, An Arrangement Calculus, Its Complexity, and Algorithmic Properties, KI 2003, LNAI 2821, pp. 580–590, 2003.
5. C.L. Weng, X.D. Lu, and Q.N. Deng, Formalizing Service Publication and Discovery in Grid Computing Systems, GCC 2003, LNCS 3032, pp.669–676, 2004.
6. Z.W. Qi, C. Fu, D.Y. Shi, J.Y. You, and M.L. Li, Membrane Calculus: A Formal Method for Grid Transactions, GCC 2004, LNCS 3251, pp.73–80, 2004.
7. R. Milner, *Communication and Concurrency* (International Series in Computer Science, Prentice Hall, 1989).
8. R. Milner, *Communicating and Mobile Systems: the π -Calculus* (Cambridge 1999).
9. R. Milner, J. Parrow, and D. Walker, A Calculus of Mobile Processes, Part I and Part II. *Information and Computation* **100**(1), 1-77 (1992).
10. C. A. R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, 1984).
11. W.M.P Van der Aalst, Structural Characterizations of Sound Workflow Nets, Technical Reports, 96/23, Eindhoven: Eindhoven University of Technology, 1996.