RC23606 (C0505-003) May 12, 2005
Computer Science

IBM Research Report

Specifying and Verifying Business Process Models with Labeled Pi Calculus

Ke Xu[1], Ying Liu[2], Cheng Wu[1]

[1]Automation Department
Tsinghua University
Beijing 100084
China

[2]IBM Research Division
China Research Laboratory
HaoHai Building, No. 7, 5th Street
ShangDi, Beijing 100085
China

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Specifying and Verifying Business Process Models with Labeled Pi Calculus[1]

Ke Xu[(1)*], Ying Liu[(2)*], Cheng Wu[(1)]

[(1)] Automation Department, Tsinghua University, Beijing 100084, China
[(2)] IBM China Research Lab, Haohai Building, Beijing 100085, China

Abstract. Because of the strong expressive power and analysis capabilities, Pi calculus has been applied in the business process modeling area by more and more people. In existing works, Pi calculus is often used to capture the action execution logic of a business process. But the real situation is that some data and resource information are also critical for business process models besides action execution logic. Although it is possible to indirectly specify the behavior of data and resource with Pi processes, it is not intuitive for understanding and analyzing the holistic semantics of a business process. An extension for Pi calculus with state information, called labeled Pi calculus, is proposed in this paper. The main idea of labeled Pi calculus is to replace the abstract state name with the detailed state information, so that it can not only specify the semantics of action execution logic but also describe state transition of a business process. It can be regarded as an integrated formal method of state-based paradigm and event-based paradigm. Because state information is integrated to Pi calculus in a loosely coupled way, the theory and analysis capabilities of Pi calculus can be well-preserved in labeled Pi calculus. Moreover, new state simulation and equivalency relations are defined for labeled Pi calculus, which improve the analysis capability of Pi calculus considerably. A business process example is used to illustrate how to formalize the semantics of a business process with labeled Pi calculus and how to analyze the properties of it.

Key Words: Labeled Pi Calculus; State Equivalency; Hybrid Transition System

1. Introduction

Model-driven architecture (MDA)[15] is gaining more and more acceptance from both field and academic sides because of its value in accelerating and rationalizing business application development. Business process modeling plays an important role in MDA. In MDA, process models are used to capture the real business operations as platform independent models. The fidelity checking of a business process model is crucial for helping improve the feasibility of MDA in real business cases. Simulation

[1] There is a possible confusion about the usage of the term 'label' in labeled transition systems (LTS) and the state labeling function in Kripke Structure. The term 'label' used in this paper rather refers to the latter one. Therefore to avoid this confusion, we will call a transition 'labeled' by action *a* in LTS to be a transition 'fired' by action *a* in this paper.

* Contact Author. Phone Number: 086-13810103537; E-mail: xk02@mails.tsinghua.edu.cn
* Contact Author. Phone Number: ; E-mail: aliceliu@cn.ibm.com

technologies have been widely applied to business process modeling tools to predicate the behavior and performance of a business process model. However, the reliability of simulation technologies is always a question due to its random character.

Model checking technologies have been recognized in protocol verification and hardware design verification. How to apply model checking technology to the area of process oriented management systems is a hot topic in recent years. In fact, there have some existing achievements in applying formal methods into complex process modeling systems, such as grid workflow and various business process modeling tools. Though there are some arguments about what is the best theoretical foundation of business process management, petri Net[6] and Pi calculus[1][2] are regarded as two promising candidates. The benefits of theoretical foundation are obvious, and the following two points are most important: one is that a solid mathematical method can help define the unambiguous semantics of a process model, therefore, it can be translated into an unambiguous implementation; the second point is that the mature analysis capability of a mathematical model can help analyze its properties.

Pi calculus put forward by Robin Milner[1] has attracted a lot of attentions, and a variety of research works have been done to apply Pi calculus in the field of process modeling[3], service composition[4][5], etc. The reason of the wide recognition for Pi calculus in business process management is because Pi calculus has three native features: simple syntax and semantics but with high expressive power; the mobility feature which makes Pi calculus specialized in capturing the dynamic interaction among system components; strong analysis capability, such as bi-simulation and model checking supporting. Despite all these merits, applying Pi calculus to business process management is still a big challenge because of the following problems:

1. Pi calculus is event-based formal method, i.e., transition is the first-class but without explicit state information. On the contrary, the execution logic of actions is only one aspect in business process models. Input and output data information as well as resources and their status are also needed to be explicitly modeled. How to completely formalize a business process model with Pi calculus is a problem.
2. Similarly to the first problem, properties that can be accepted by model checking tools of Pi calculus are often action related properties because Pi calculus is event-based method. However, the reality is that our customers really don't want to differentiate event-based properties and state-related properties. An example is give here, 'each *assemble product* activity must be carried out before the *factory directive* has approved the activity and the needed machines are *free*', which is a hybrid property including both action execution order and state confirmation.
3. The computation complexity of model checking is always a concern.
4. In order to using analysis capability of Pi calculus, users are demanded to specify the required properties in formal formulas since almost all currently available model checking tools accept only logical formulas as input for properties specification. However, the reality is that business people usually don't want to care logical formulas.

In this paper, we will focus on the first two problems and provide some clues for the third one. For the forth problem, we will propose to design a pattern-based property specification language for supporting easily writing properties. This work will be discussed in a separate paper.

The main question is that the specification and analysis power of traditional Pi calculus is not enough in analyzing a process when it is interpreted in a state-based view. Although there is a recent trend in the integration of state/event based formalization and specification language[16-19], there is still much work left to be done to address the above problems since the most frequently used existing formal techniques in business process modeling are purely event-based (e.g. process algebras) or state-based (e.g. finite state machine and Petri nets). Besides, as has been shown in [14] and [18], the hybridization of actions and states will make both the property specification and modeling of systems to be more effective. Therefore, in this paper we propose to extend Pi calculus to support both state and event based specification and verification of process models. Our idea can be concluded in the following three points: to extend Pi calculus with state–based feature; to exploit bi-simulation not only in checking behavioral equivalency but also state equivalency; to reflect different user concerns in Pi calculus so that effective reduction techniques can be developed and the size of the state space can be well managed for decreasing computation complexity.

In this paper, a Pi calculus extension, namely, labeled Pi calculus will be introduced how to solve the above problems. There are three basic concepts in labeled Pi calculus, *names, processes*, and *labels*. The semantics of two concepts, *names* and *processes*, are inherited from Milner's Pi calculus. The new concept, *label*, is borrowed from traditional Kripke structure. The labeled Pi calculus is built on the Pi calculus to extend its capability in specifying the state information of a system in an explicit and simple way. A transition system which is a combination of labeled transition system (LTS) and traditional kripke structure is used to interpret the semantics of labeled Pi calculus. As a result, labeled Pi calculus can provide three different perspectives: the state-based perspective for showing user concerned business propositions, the event-based perspective for showing behavioral information of business processes, and the hybrid perspective of the above two. Consequently, all the existing Pi calculus related logics[7][8], traditional CTL/LTL like temporal logics and hybrid state/event based logics[14][18] can be used for the verification of labeled Pi calculus from different perspective. The syntax and semantics of labeled Pi calculus will be fully introduced in this paper. Besides, the concepts of hybrid bisimulation and state bisimulation, which are theoretical generalizations of similar relations in [9] for the purpose of process structural optimization, are also discussed as basic analysis capability of the labeled Pi calculus.

To our best knowledge, there has been some previous works on incorporating the dynamic event-based Pi calculus with a structured state-based formalism. Ciobanu and Rotaru[10] attempted to define a state-based machine-like formalism for the Pi calculus; Kenji, Jin and Gabriel[11] integrated the state transition semantics of Object-Z and Pi calculus reduction rules. The result of their work is a new specification language called *PiOZ*. Our work differs from the above works at least from the following three aspects:

1. From theoretical aspect, our work is a direct extension of the theory of Pi calculus, instead of the semantic encoding of the Pi calculus into other languages;
2. From practical aspect, our work provides a rather simple and pluggable mechanism for users to specify their concerned system information, such that not only the core theory and verification techniques of event-based Pi calculus

remains intact, but also new state-based information and verification techniques can be effectively exploited for system analysis;

3. In *PiOZ*, only monadic version of Pi calculus[1] is supported while labeled Pi calculus works well in the polyadic form[12].

This paper is organized as follows. In the next section, the concept and properties of labels and labelsets are formally introduced for the understanding of the labeled Pi calculus. In section 3, the theoretical framework of Milner's Pi calculus is extended to integrate both action and state information in labeled Pi calculus. The expansion semantics and extended transition semantics for labeled Pi calculus are also introduced. A process example is formalized with labeled Pi calculus for helping understand the semantics and application of labeled Pi calculus in section 4. In section 5, state information is integrated into the open bisimulation in Pi calculus. Besides, new state simulation relations are also introduced for the analysis of model equivalencies from a pure state-oriented perspective. State simulation can be viewed as a complement to the behavioral bisimulation analysis in Pi calculus. The connections between the newly proposed relations with open bisimulation are also analyzed. Conclusions and future works of this paper can be found in the last section.

2. Preliminaries

'Labels' is an extra basic entity introduced in Labeled Pi calculus aside from 'names' and 'processes'. Before diving into the details of labeled Pi calculus, the concepts and semantics of labels and labelsets are discussed in this section.

2.1 Syntax of Labels and Labelsets

Definition 2.1 (Label): A label is a triple *(lname, Act, cvalue)*, where

- *lname* is the name of this label;
- *Act* is a set of pairs, and each pair is denoted as *(act, op)*, where *act* is an action defined in Pi calculus that may change the label's value and *op* is the operation which acts on the value of the label when action *act* occurs;
- *Cvalue* is the current value of the label.

The *Act* in the definition of label can be regarded as a mapping from the actions to the update of corresponding label values.

Definition 2 gives the definition of a labelset.

Definition 2.2 (Labelset): A labelset is a pair, denoted as *(lsname, labels)*, where

- *lsname* is the name of this labelset;
- *labels* is a set of all labels, denoted as $labels = \{label_1; ...; label_n\}$.

The detailed syntax is given as below.

$$labelset ::= label \mid labelset; label$$
$$label ::= lname^{\{cvalue\}} : pola(PolarityExpr) \mid lname^{\{cvalue\}} : var(VariableExpr)$$
$$PolarityExpr ::= a\ \ PolarityOp \mid PolarityExpr, PolarityExpr$$
$$VariableExpr ::= a\ VariableOp \mid VariableExpr, VariableExpr$$
$$PolarityOp ::= + \mid - \mid /$$
$$VariableOp ::= -> value$$

In the above syntax, *lname* denotes the name of a label and *a* denotes an action which has the same semantics as that of action in Pi calculus. Label has two types, *polarity label* and *variable label,* which are differentiated with different keywords, *pola* and *var*. Different types of labels have different kinds of available operations '*op*'. Informally, *polarity label*s are labels with integer values which are increased, decreased and reset to zero by operation '+', '-' and '/' respectively. *variable label*s are labels with string values which are assigned by operation '->*value*'. *Cvalue* denotes the current value of the label. In default, it is zero for *polarity label*s and '*Null*' for *variable label*s if it is not explicitly specified in label definition. *Cvalue* will be updated through system transitions fired by corresponding action *a*.

The semantics of *op* is defined as below with a function, *Compute()*.

$$Compute : op \times cvalue \rightarrow cvalue$$
$$Compute(+, cvalue) = cvalue + 1 \qquad Compute(-, cvalue) = cvalue - 1$$
$$Compute(/, cvalue) = 0 \qquad Compute(\rightarrow v, cvalue) = v$$

As an example, labelset '*my1*' composing of two labels, '*DSCapacity*' and *'ConnTrans'*, is defined as following. Label '*DSCapacity*' is a *polarity label* which indicates that there is a concerned property of 'DataStore Capacity' whose value will be increased by an input action '*put*' and decreased by an input action '*get*'. On the other hand, label '*ConnTrans*' is a '*variable*' label which may indicate that the currently connected transmitter is from '*DeutscheTelekom*' (abbreviated to '*DTele*') if a transition is made by output prefix '*talkDT*' in Pi calculus.

$$LabelSet\ \ my1 =_{def} DSCapacity^{\{0\}} : pola(put+, get-); ConnTrans^{\{Bell\}} : var(\overline{talkDT} -> DTele)$$

Therefore, $DSCapacity = (DSCapacity, Act_1, 0)$ where $Act_1 = \{(put, +), (get, -)\}$ and $ConnTrans = (ConnTrans, Act_2, Bell)$, where $Act_2 = \{(\overline{talkDT}, DTele)\}$.

2.2 The update of labels

As mentioned in 2.1, the value of a label will be updated by the actions of its definition. *Ch(a)* is a function defined to get the prefix of an action *a*:

$$Ch(a) = \begin{cases} \underline{x} & if\ \ a = x(y) \\ \overline{x} & if\ \ a = \overline{x} < y > \\ \tau & if\ \ a = \tau \end{cases}$$

Definition 2.3: Two actions, *a1* and *a2,* are matched, denoted as $a1\ ;\ a2$, IFF the following two conditions are satisfied:

(1) $a1 \neq \tau\ \wedge\ a2 \neq \tau$; (2) $Ch(a1) = Ch(a2)$.

Given an action a and a label $l=(lname, Act, cvalue)$, where $Act_a = \{act \mid (act, op) \in Act \ \wedge \ act \ ; \ a\}$ and $Op_a = \{op \mid (act, op) \in Act \ \wedge \ act \ ; \ a\}$, the definition of well-formedness of label is given as follows.

Definition 2.4: A label $l=(lname, Act, cvalue)$ is said to be well-formed IFF for each $a \in Act$, $| Act_a | = 1$.

Definition 2.4 implies that each action prefix contained in Act of a well-formed label should be unique. Therefore we have $|Op_a|=1$ and $(act, op) \in Act$ if $act \in Act_a$ and $op \in Op_a$ for well-formed labels.

For example, the label *DSCapacity* and *ConnectedTrans* in the previous section are well formed. The followings are two label examples which are not well formed.

label $DSCapacity^{\{0\}}$ ' : $pola(put+, get-, put-)$

label $ConnTrans^{\{BJTele\}}$ ' : $var(\overline{talkDT->DTele}, \overline{talkDT->AT \ \& \ T})$

In the rest of the paper, the term 'label' will all refer to 'well formed label'.

For label $l=(lname, Act, cvalue)$, its current value will be updated after a transition is fired by action a. The update of the current value for a label caused by action a is explained with a update() function as follows:

$$update : cvalue \times action \rightarrow cvalue$$

$$update(cvalue, a) = \begin{cases} cvalue & | Act_a | = 0 \\ cvalue + 1 & a \in | Act_a | \ \wedge \ Op_a = \{+\} \\ cvalue - 1 & a \in | Act_a | \ \wedge \ Op_a = \{-\} \\ 0 & a \in | Act_a | \ \wedge \ Op_a = \{/\} \\ v & a \in | Act_a | \ \wedge \ Op_a = \{->v\} \end{cases}$$

Consequently, after a transition is fired by action a, the label l is updated to $l_{update(a)} = (lname, Act, cvalue')$ where $cvalue' = update(cvalue, a)$. If the value of label l is updated by a sequence of action $a_1, ...a_n$, it is denoted as $update(cvalue, a_1, ...a_n)$:

$$update(cvalue, a_1, ...a_n) = update(...update(update(cvalue, a_1), a_2)..., a_n) .$$

Furthermore, $l_{update(a_1,...,a_n)}$ and $\{l_1, l_2, ...\}_{update(a_1,...,a_n)}$ can also be used to denoted the updated label and labelset caused by a sequence of actions $a_1, ...a_n$ respectively where $l_{update(a_1,...,a_n)} = (lname, Act, update(cvalue, a_1, ...a_n))$ and $\{l_1, l_2, ...\}_{update(a_1,...,a_n)} = \{l_{1update(a_1,...,a_n)}, l_{2update(a_1,...,a_n)}, ...\}$.

2.3 Label Composition and Reduction

Since a labelset essentially is the composition of a set of labels, $Labelset\ lsname \stackrel{def}{=} label_1; ...; label_n$, it is necessary to define its behavior for a new label $label_i$ $(i \neq 1, ..., n)$ being inserted into the labelset. In this section, different relations

between two labels are firstly defined, and then the reduction rules of labelset based on its well-formed definition will be introduced.

Definition 2.5: For two labels: $l_1 = (lname_1, Act_1, cvalue_1)$ and $l_2 = (lname_2, Act_2, cvalue_2)$,

- l_1 is said to be **irrelevant, denoted as** l_2 ($l_1 \_ ' l_2$), if $lname_1 \neq lname_2$;
- l_1 is said to be **overlapping, denoted as** l_2 ($l_1 \_ '' l_2$), if $lname_1 = lname_2$, and for a new label $l = (lname_1, Act_1 \cup Act_2, cvalue_1)$, l is well formed;
- l_1 is said to be **conflicting, denoted as** l_2 ($l_1 \_ ''' l_2$), if $lname_1 = lname_2$, and for a new label $l = (lname_1, Act_1 \cup Act_2, cvalue_1)$, l is not well formed

Similar to the definition of well-formed label, definition 2.6 defines the well-formed labelset.

Definition 2.6: For a labelset *ls=(lsname, labels),* where $labels = \{label_1; ...; label_n\}$, is said to be well-formed if for any $label_i$ and $label_j$ ($i \neq j$) contained in *labels*, $label_i$ and $label_j$ are irrelevant.

For any non-well-formed labelset, reduction should be carried out to make it a well-formed labelset. Given a well-formed labelset *ls=(lsname, labels)* where $labels = \{l_1; ...; l_n\}$ and a label *l=(lname, Act, cvalue)*, their composition forms a new labelset $ls_c = (lsname, labels_c)$ where $labels = \{l_1; ...; l_n; l\}$. Here are three reduction rules for labelset ls_c which will make it to be well-formed:

$$Hold \quad \frac{\{l_1; ...; l_n; l\} \quad l \_ ' l_i}{\{l_1; ...; l_n; l\}} \quad i = 1, ..., n \tag{1}$$

$$Supplement \quad \frac{\{l_1; ...; l_n; l\} \quad l \_ '' l_i}{\{l_1; ...; l_i; ...; l_n\} \quad l_i = (lname_i, Act_i \cup Act, cvalue_i)} \tag{2}$$

$$Ignore \quad \frac{\{l_1; ...; l_n; l\} \quad l \_ ''' l_i}{\{l_1; ...; l_n\}} \tag{3}$$

Hold rule indicates an *irrelevant* label can be directly inserted into the labelset, *Supplement* rule indicates an *overlapping* label in the labelset needs to be updated by a unite of the *Act* sets, and *Ignore* rule indicates a label will be discarded if it *conflicts* with a label in the labelset.

Consequently, if we write $ls_1, ..., ls_n$ in short for $labelset_1, ..., labelset_n$, a reduction function $LRed(ls_1, ls_2)$ can be implemented based on Rule (1)-(3) for the composition of different *labelset*s. Moreover, the reduction function $LRed(ls_1, ..., ls_n)$ is formally defined as:

$$LRed(ls_1, ..., ls_n) = LRed(...LRed(LRed(ls_1, ls_2), ls_3), ..., ls_n) \tag{2.3-1}$$

Using the above (2) and (3), two labelset *my1* and *my2* are composed to a new labelset in the following example.

$$LabelSet\ my1 =_{def}\ DSCapacity^{\{0\}} : pola(+put, -get); ConnTrans^{\{BJTele\}} : var(\overline{talk} -> DTele)$$

$$LabelSet\ my2 =_{def}\ DSCapacity^{\{99\}} : pola(/\ reset); ConnTrans^{\{BJTele\}} : var(\overline{talk} -> AT\ \&\ T)$$

$$LRed(my1, my2) = DSCapacity^{\{0\}} : pola(+put, -get, /\ reset); ConnTrans^{\{Bell\}} : var(\overline{talk} -> DTele)$$

The reduction function *LRed* can be proved to satisfy the following property.

Lemma 2.1: $LRed(ls_1, ls_2, ls_3) = LRed(LRed(ls_1, ls_2), ls_3) = LRed(ls_1, LRed(ls_2, ls_3))$

Proof: The equation $LRed(ls_1, ls_2, ls_3) = LRed(LRed(ls_1, ls_2), ls_3)$ can be directly conducted from the definition (2.3-1). To prove that

$$LRed(ls_1, ls_2, ls_3) = LRed(ls_1, LRed(ls_2, ls_3)) \quad (2.3\text{-}2)$$

Suppose l_{1i} , l_{2i} and l_{3i} to be an arbitrary label contained in ls_1 , ls_2 and ls_3 respectively,

if $l_{3i} \_ ' l_{2i}$, equation (2.3-2) obviously holds;

if $l_{3i} \_ ''' l_{2i}$, equation (2.3-2) holds because the effect of l_{3i} is simply ignored;

if $l_{3i} \_ '' l_{2i}$ and $l_{2i} \_ ' l_{1i}$, equation (2.3-2) obviously holds;

if $l_{3i} \_ '' l_{2i}$ and $l_{2i} \_ '' l_{1i}$, equation (2.3-2) holds because:

$$(l_{1i}, (Act_{1i} \cup Act_{2i}) \cup Act_{3i}, cvalue_{1i}) = (l_{1i}, Act_{1i} \cup (Act_{2i} \cup Act_{3i}), cvalue_{1i}) ;$$

if $l_{3i} \_ '' l_{2i}$ and $l_{2i} \_ ''' l_{1i}$, equation (2.3-2) holds because the effect of both l_{2i} and l_{3i} is simply ignored. □

Corollary 1: A more general property holds for *LRed* based on Lemma 1:

LRed(ls1,…,lsn)= LRed(ls1,…, LRed(lsi,lsi+1),…,lsn) where *i=1,…,n-1* (2.3-3)

Corollary 2: $LRed(ls_1, ls_2) \neq LRed(ls_2, ls_1)$.

The following two items define the preorder relation between two labelsets, ls_1 and ls_2 :

- ls_1 p ls_2 , if for each label l_1*=(lname1, Act1, cvalue1)* in ls_1, there is a label l_2*=(lname2, Act2, cvalue2)* in ls_2 such that $lname_1 = lname_2$ and $cvalue_1 = cvalue_2$ are hold.
- $ls_1 = ls_2$, if both ls_1 p ls_2 and ls_2 p ls_1 are hold.

Lemma 2.2: $\forall ls'$, ls p $LRed(ls, ls')$.

Proof: From reduction rules (1)-(3), we have (1) $ls = LRed(ls, ls')$ if for each label l in *ls'*, rule '*Supplement*' or '*Ignore*' is applied during the composition; (2) ls p $LRed(ls, ls')$ if there exists a label l in *ls'*, such that rule '*Hold*' is applied during the composition. □

Lemma 2 illustrates an important conclusion that label composition and reduction forms a preorder between labelsets.

2.4 Label substitution

Just like the substitution of names in Pi calculus, we denote l^{σ} to be a label substitution, where l is a label and $\sigma = \{\%/x, \%/y\}$.

For label *l=(lname, Act, cvalue)* and $l^{\sigma} = (lname, Act^{\sigma}, cvalue)$ where Act^{σ} is defined as follows:

for each $(act, op) \in Act$, $Ch(a)$ is substituted to a new name denoted by $Ch(a)_{\sigma}$ if $Ch(a) \in \mathrm{cod}(\sigma)$ where $\sigma = \{\tilde{x}/\tilde{y}\}$ and $\mathrm{cod}(\sigma) = \{\tilde{y}\}$.

Similarly, denote $\{l_1, l_1, ...\}^{\sigma}$ to be a labelset substitution where $\sigma = \{\tilde{x}/\tilde{y}\}$, $\{l_1, l_1, ...\}^{\sigma} = \{l_1^{\sigma}, l_2^{\sigma}, ...\}$.

3. Labeled Pi Calculus

As the concept and properties of labels and labelsets have been introduced in the previous section, the syntax and semantics of labled Pi calculus will be discussed in this section.

3.1 Extended Syntax of Labeled Pi Calculus

The labeled Pi calculus is built on the polyadic version of Pi calculus[12]. Its syntax is given as below.

$P ::= \sum_{i=1}^{n} a_i.P_i \mid (new\ x)P \mid !P \mid P \mid Q \mid \phi P \mid A(y_1, ..., y_n)\{labelset\} \mid 0$

$a_i ::= \bar{x} < \tilde{y} > \mid x(\tilde{y}) \mid \tau$

$\phi ::= [x = y] \mid \phi \wedge \phi \mid \neg\phi$

Any process P that is defined by process identifier $A(y_1, ..., y_n)\{labelset\}$ in the above syntax is called a labeled process. The only extension to Pi calculus is the introduction of a new operator '{}'. With this operator, a *labelset* can be explicitly associated with a process identifier which essentially differs a *labeled Pi process* with a regular process in Pi calculus.

In a labeled Pi process, it is possible that such an association is redundant because some actions contained in a label can never be used. Before defining the redundancy of an association in a label, we first introduce the notion of *Observable*.

Definition 3.1 [2]**:** A name x and its co-name $\bar{x}$ are observable to process P, denoted by $P \downarrow_x$ and $P \downarrow_{\bar{x}}$ respectively, if x is a free name of P ($x \in fn(P)$).

Definition 3.2: A labeled Pi process, defined by $A(y_1, ..., y_n)\{labelset\}$ where $labelset = (lsname, \{l_1; ...; l_n\})$ with $l_i = (lname_i, Act_i, cvalue_i)$, is called to be ***minimal*** IFF for each $(a, op) \in Act_i$ $(i = 1, ..., n)$, $A \downarrow_{Ch(a)}$. Otherwise A is ***Redundant*** because of the invalid association of (a, op), which is denoted as $A \not\in (a, op)$.

[2] The definition is modified from 'Observability predicates' in [12]. Such a modification is necessary since free names as name parameters are also potential for changing the value of a label.

Each labeled Pi process should be minimal such that the value change of its associated labels can be only caused by actions that are 'observable' from the outside of the process. Based on definition 3.2 we can now provide the fourth reduction rule to make a labeled process to be minimal. It implies that all invalid associations of actions to labels will simply be removed.

$$Abandon \quad \frac{A(y_1,...,y_n)\{ls\} \qquad A \notin (a,op) \qquad \forall l_i \in ls, l_i = (lname_i, Act_i, cvalue_i)}{A(y_1,...,y_n)\{ls'\} \qquad \forall l_i' \in ls', l_i' = (lname_i, Act_i \setminus \{(a,op)\}, cvalue_i)} \quad (4)$$

Definition 6 together with Rule (4) suggests that restricted names in a process cannot be explicitly used to change the label value associated with the process, and such a label change is deemed as invisible for the process. In the real applications, it means that some states in the sub-component are hidden from the super-component of a system, such that the super-component need not care and will never change these hidden states.

Supposing $\partial(P) = \{l_1, l_2, ...\}$ to be the labelset associated with process P, $\partial(P)$ is empty if P is not explicitly associated with any labelset in its definition. The following propositions can be used to determine the labelset associated with each labeled Pi process.

(1) $A(y_1,...,y_n)\{ls\} =_{def} P2$,,where $ls = (lsname, \{l_1;...;l_n\})$, $\partial(A(y_1,...,y_n)\{ls\}) = \{l_1';.....;l_n'\} \cup \partial(P2)$ where $\{l_1';.....;l_n'\}$ is reduced from $\{l_1;...;l_n\}$ by rule (4);

(2) $A(y_1,...,y_n) =_{def} P2, \partial(A(y_1,...,y_n)) = \partial(P2)$ (3) $\partial(0) = \{\}$

(4) $\partial(a.P) = \{\}$; (5) $\partial(\phi P) = \partial(P)$; (6) $\partial(P+Q) = \partial(P) \cup \partial(Q)$;

(7) $\partial(P \mid Q) = \partial(P) \cup \partial(Q)$; (8) $\partial((new\ x)P) = \partial(P)$; (9) $\partial(P_\sigma) = \partial(P)^\sigma$;

Table 1. Propositions for Labelset Association

In table 1, proposition 1-2 indicate that only a process directly defined by a process identifier can possibly be associated with labelsets. Otherwise the labelset will be empty. Proposition 3-9 are used to determine the associated labelsets when different operators in Pi calculus are met.

For example, consider $TestP(put, talk1)\{my1\} =_{def} (!\, put(x).\overline{talk1} < x > .0)\{my1\}$ where the labelset *my1* is defined in the previous section. $\partial(TestP(put, talk1)\{my1\}) = \{DSCapacity^{\{0\}} : pola(put+); ConnTrans^{\{Bell\}} : var(\overline{talk} - > DTele)\}$,where the association of action *get* to *DSCapacity* is removed since it is redundant.

3.2 Hybrid Transition System

The traditional labeled transition system is the basis for interpreting the behavior of Pi calculus.In this paper, we use hybrid transition system to interpret the semantics of labeled Pi calculus. The hybrid transition system is firstly introduced here.

Definition 3.3[3]: A hybrid transition system $(S, M, \{\xrightarrow{a} \mid a \in M\}, L)$ consists of a set S of states, a set M of transition labels, a set of transitions $\xrightarrow{a} \subseteq S \times S$ for each $a \in M$ and a state labeling function $L : S \to Labelset$ that associates each state with a set of labels.

In fact, the hybrid transition system can be regarded as an integration of the features between the labeled transition system and the Kripke Structure, such that a set of transitions is considered instead of a set of relations between states. A transition in the hybrid transition system is denoted as $(P \hookrightarrow Ls) \xrightarrow{a} (Q \hookrightarrow Ls')$ which has the following semantics:

An old state P which is associated with a set of labels Ls, denoted as $P \hookrightarrow Ls$ *, transits through an action a to a new state Q and meanwhile the set of labels is updated to Ls' for the new state, denoted as* $Q \hookrightarrow Ls'$.

The difference between the labelset *Ls* in *(P, Ls)* and $\partial(P)$ is that *Ls* represents the labelset associated with a specified state of the system, but $\partial(P)$ represents the labelset associated with a specified Pi process. Therefore, the labels in *Ls* can be viewed as all global propositions which are asserted to be true in a state of the system and the labels in $\partial(P)$ can be viewed as new possible labels which may contribute to the expansion of current labels in *Ls*. It will be more clearly defined in section 3.3 and 3.4. To avoid their confusion, we will call *Ls* to be *state labelset* and $\partial(P)$ to be *process labelset*.

3.3 Transition semantics for labels

Labeled Pi calculus is an extension of Pi calculus with label and labelset. How to understand the semantics of label and labelset will be critical. Labeled transition system is often used to interpret the semantics of Pi calculus. In fact, the intuitive understand for labeled Pi calculus is that the name of each state in Pi calculus is replaced with some propositions. If users don't want to care the detailed state information, it is can be easily abstracted with a state name. Just because of the flexible mechanism, the expressive power and analysis capabilities of Pi calculus, such as bisimulation theory and model checking method, are well preserved in labeled Pi calculus. In order to make full use of this advantage, the principle of integrating the semantics between Pi calculus and label should be in a loosely coupled way.

Therefore, the reduction rules and the structural congruence of labeled Pi calculus will remain the same as that of Pi calculus. But, transition rules need some modifications in order to define the semantics about label updating due to process transition.

$$OUT \quad \frac{}{(\bar{x}<y>.P \hookrightarrow Ls) \xrightarrow{\bar{x}<y>} (P \hookrightarrow Ls_{update(\bar{x})})} \qquad INP \quad \frac{}{(x(z).P \hookrightarrow Ls) \xrightarrow{x(z)} (P \hookrightarrow Ls_{update(x)})}$$

[3] A similar concept has also been proposed in [14], which is called 'Labeled Kripke Structure' instead in their work.

$$TAU \quad \frac{}{(\tau.P \hookrightarrow Ls) \xrightarrow{\tau} (P \hookrightarrow Ls)} \qquad MAT \quad \frac{(P \hookrightarrow Ls) \xrightarrow{a} (P' \hookrightarrow Ls_{update(a)})}{([x=x]P \hookrightarrow Ls') \xrightarrow{a} (P' \hookrightarrow Ls'_{update(a)})}$$

$$SUM-L \quad \frac{(P \hookrightarrow Ls) \xrightarrow{a} (P' \hookrightarrow Ls_{update(a)})}{(P+Q \hookrightarrow Ls') \xrightarrow{a} (P' \hookrightarrow Ls'_{update(a)})}$$

$$PAR-L \quad \frac{(P \hookrightarrow Ls) \xrightarrow{a} (P' \hookrightarrow Ls_{update(a)})}{(P \mid Q \hookrightarrow Ls') \xrightarrow{a} (P' \mid Q \hookrightarrow Ls'_{update(a)})} \quad bn(a) \cap fn(Q) = \varnothing$$

$$COMM-L \quad \frac{(P \hookrightarrow Ls) \xrightarrow{\bar{x}<y>} (P' \hookrightarrow Ls_{update(\bar{x})}) \;\; (Q \hookrightarrow Ls') \xrightarrow{x(z)} (Q' \hookrightarrow Ls'_{update(x)})}{(P \mid Q \hookrightarrow Ls'') \xrightarrow{\tau} (P' \mid \{y/z\}Q' \hookrightarrow Ls''_{update(\bar{x},x)})}$$

$$CLOSE-L \quad \frac{(P \hookrightarrow Ls) \xrightarrow{\bar{x}<z>} (P' \hookrightarrow Ls_{update(\bar{x})}) \;\; (Q \hookrightarrow Ls') \xrightarrow{x(z)} (Q' \hookrightarrow Ls'_{update(x)})}{(P \mid Q \hookrightarrow Ls'') \xrightarrow{\tau} (new\ z)(P' \mid Q' \hookrightarrow Ls''_{update(\bar{x},x)})}$$

$$RES \quad \frac{(P \hookrightarrow Ls) \xrightarrow{a} (P' \hookrightarrow Ls_{update(a)})}{((new\ z)P \hookrightarrow Ls') \xrightarrow{a} ((new\ z)P' \hookrightarrow Ls'_{update(a)})} \quad z \notin n(a)$$

$$OPEN \quad \frac{(P \hookrightarrow Ls) \xrightarrow{\bar{x}<z>} (P' \hookrightarrow Ls_{update(\bar{x})})}{((new\ z)P \hookrightarrow Ls') \xrightarrow{\bar{x}<z>} (P' \hookrightarrow Ls'_{update(\bar{x})})} \quad z \neq x$$

$$REP-ACT \quad \frac{(P \hookrightarrow Ls) \xrightarrow{a} (P' \hookrightarrow Ls_{update(a)})}{(!P \hookrightarrow Ls') \xrightarrow{a} (P' \mid !P \hookrightarrow Ls'_{update(a)})}$$

$$REP-COMM \quad \frac{(P \hookrightarrow Ls) \xrightarrow{\bar{x}<y>} (P' \hookrightarrow Ls_{update(\bar{x})}) \;\; (P \hookrightarrow Ls') \xrightarrow{x(z)} (P'' \hookrightarrow Ls'_{update(x)})}{(!P \hookrightarrow Ls'') \xrightarrow{\tau} ((P' \mid \{y/z\}P'') \mid !P \hookrightarrow Ls''_{update(\bar{x},x)})}$$

$$REP-CLOSE \quad \frac{(P \hookrightarrow Ls) \xrightarrow{\bar{x}<z>} (P' \hookrightarrow Ls_{update(\bar{x})}) \;\; (P \hookrightarrow Ls) \xrightarrow{x(z)} (P'' \hookrightarrow Ls_{update(x)})}{(!P \hookrightarrow Ls') \xrightarrow{\tau} (((new\ z)(P' \mid P'') \mid !P) \hookrightarrow Ls'_{update(\bar{x},x)})}$$

Table 2. Extended Transition Rules in Labeled Pi Calculus

In table 2, the semantics of the notion for labelset update can be referred in section 2.2. Besides, except for rule *MAT*, *SUM-L*, *PAR-L* and *RES*, in the above rules only the action prefixes x and $\bar{x}$ (which are also actions themselves with τ to be empty) are considered in the update of labels. This is because only action prefixes are needed to match different actions as defined in Definition 2.3.

The above transition rules are based on the late transition semantics of Pi calculus[12]. As can be seen in table 2, these rules essentially preserve all the transition semantics of Pi calculus and meanwhile define how the values of the labels in state are updated.

3.4 Expansion semantics for labels

The previously defined transition semantics show how the value of state labelset is updated by different types of transition rules. However, as stated in section 3.1 and 3.2, process evolves during system transitions such that new process labelset may emerge. This is actually one of the advantages of labeled Pi calculus that hidden labels can be unveiled when different processes are entered and they will be merged to become part of the state labelset. Look at the following example:

$$Labelset\ ls1 = l1^{\{0\}} : pola(a+) \qquad Labelset\ ls2 = l2^{\{null\}} : var(t->ok)$$

$$P(a,b)\{ls1\} =_{def} (new\ t(a(x).Q(b,t,x)\{ls2\}))\{ls1\}$$

To understand the transition of process *P*, first the labelset *ls1* is recognized by propositions in table 1. A transition *a(x)* is then fired and *ls1* is updated according to the transition semantics. At this point, a new labelset *ls2* emerges which contains a new label *l2*. In fact, *l2* is a redundant for process *P* according to *Abandon* rule in section 3.1. This label, while is hidden from *P*, is visible for process *Q*. Therefore, it is necessary to define the semantics of the state labelset is expanded to accept these new labels.

To determine how new labels are identified and inserted into state labelset, the following expansion rules for state labelset are defined.

Suppose the process representing the current state is *P*, the labelset of the current state before and after expansion are *Ls* and *Ls'* respectively, the expansion rules are of the following form: $P \times Ls \to Ls'$.

$LE-DEF \quad P \times Ls \to LRed(Ls, \partial(P))$	$LE-SUM \quad P1+P2 \times Ls \to LRed(Ls, \partial(P1), \partial(P2))$
$LE-SUB \quad P_\sigma \times Ls \to LRed(Ls, \partial(P)^\sigma)$	$LE-COMM \quad P1 \mid P2 \times Ls \to LRed(Ls, \partial(P1), \partial(P2))$
$LE-RES \quad (new\ x)P \times Ls \to LRed(Ls, \partial(P))$	$LE-MAT \quad \phi P \times Ls \to LRed(Ls, \partial(P))$
$LE-PREFIX \quad a.P \times Ls \to Ls$	$LE-REP \quad !P \times Ls \to LRed(Ls, \partial(P))$

Table 3. Expansion Rules in Labeled Pi Calculus

There is only one rule in tabl 3 (*LE-PREFIX*) by which the state labelset will remain unchanged. The connections between the rules in table 3 and the propositions in table 1 is that in table 3, it states the rules by which new labels are identified and inserted into the existing labelset associated with a state. On the other hand, in table 1 it defines how to exactly determine these new labels identified by rules in table 3.

However, a question left unanswered is that how the expansion semantics and transition semantics of labels are combined so that the comprehensive behavior of labels can be determined when a transition in labeled Pi calculus is committed.

Definition 3.4 (Normalized Commitment): Suppose the labelset for the initial state is got directly by function ∂. A normalized commitment of a single transition $(P \hookrightarrow Ls) \xrightarrow{a} (Q \hookrightarrow Ls')$ is of the following three steps:

A. The transition rules in table 2 are applied to update the values of the labels according to the action that occurred, and meanwhile the current state is transited to a new one.

B. One of the expansion rules in table 3 is applied *once* to identify new labels and determine the set of labels for the new state;
C. Propositions in table 1 and the reduction function *LRed* are used to help the identification and insertion of expanded labels during procedure B.

Using the above normalized commitment, the behavior of labels is fully determined by each transition steps in labeled Pi calculus. Later in the next section, this concept will be further explained in detail with concrete examples.

4. A Process Example

In this section, a concrete example, 'Product Order Handling' process, is formalized with labeled Pi calculus. The process example is shown in figure 1.

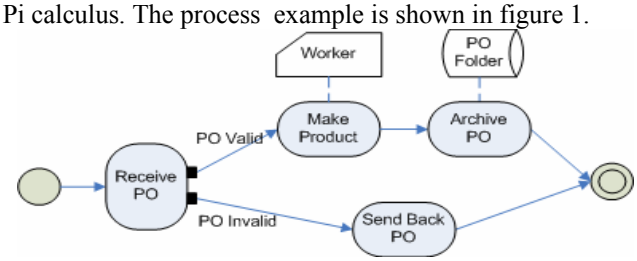

Fig. 1. A Process for Product Order Handling

This simple business process describes that a PO (Product Order) is first received and its validity is checked. If the received PO is valid, corresponding products are made by workers and afterwards the PO is archived in the PO Folder. Otherwise the PO will simply be sent back. The dotted line in figure 1 is used to indicate the used resources (workers and folders) by each activity.

Aside from the mobility feature and other advantages of Pi calculus, the event-based flow driven modeling paradigm exploited in the above process makes Pi calculus to be natural and convenient for capturing its formal behavior. There has already been some works dealing with the formalization of similar processes with Pi calculus[9][13]. On the other hand, there is also some generally useful state information which is frequently recognized in the business process modeling domain contained in the above process. The information may include: 'whether an activity is in execution?', 'Is the worker currently occupied?', 'Which role is responsible for carrying out a specified activity?', 'Is the received PO valid?', and etc. They are of great importance for helping the business analyzers to clearly understand and specify the process. Unfortunately, Pi calculus doesn't explicitly support the above state information modeling. As a matter of fact, it is also difficult for most of other existing formal methods to capture the semantics of a business process both from a behavioral perspective and state perspective.

Using the proposed labeled Pi calculus, such a process can be formalized as below. Because of the limitation of paper size, only the 'Receive PO' activity is fully formalized here as a demonstration.

$$Labelset\ ls4RecPO = RecPOInExe^{\{0\}} : pola(in+, make-, sendback-)$$

$$Labelset\ ls4Internal = POStatus^{\{Unknown\}} : var(t->valid, f->invalid)$$

$$ReceivePO(rec, make, sendback, check)\{ls4RecPO\} =_{def} (in.new\ endt\ endf\ t\ f($$

$$InternalAct(endt, endf, t, f, check)\{ls4Internal\} \mid (endt.\overline{make}.0 + endf.\overline{sendback}.0)))\{ls4RecPO\}$$

$$InternalAct(endt, endf, t, f, check)\{ls4Internal\} =_{def} (\overline{check} < t, f > .(t.\overline{endt}.0 + f.\overline{endf}.0))\{ls4Internal\}$$

The semantics of the above labeled Pi process can be understood based on the hybrid transition system in figure 2.

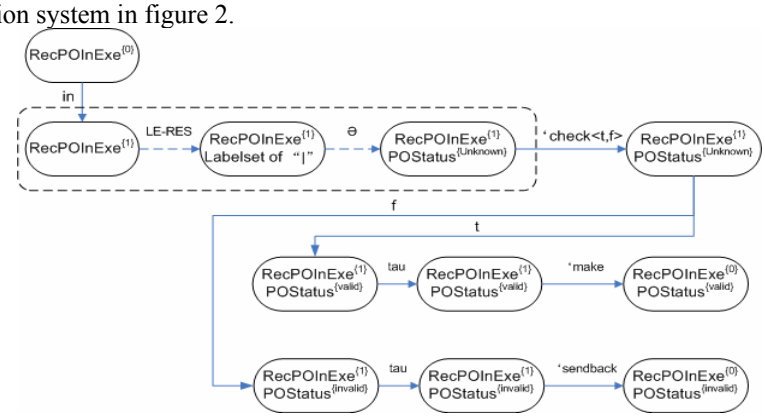

Fig. 2. Hybrid Transition System of the 'Receive PO" Activity

In figure 2, states are represented by rounded rectangles and transitions are represented by arrows. Label '*RecPOInExe*' indicates whether the activity of '*ReceivePO*' in figure 1 has started its execution; Label '*POStatus*' indicates whether the status of the received PO is *unknown*, *valid* or *invalid*.

To illustrate in detail the procedure of a normalized commitment as proposed in section 3.4, the state after the transition fired by action 'in' are decomposed into three sub-states (as circled by a dotted rounded rectangle). These sub-states reveal the construction of hybrid transition system by the semantics of labeled Pi calculus proposed in section 3.3 and 3.4. In fact, each state represented by rounded rectangle in figure 2 can be further decomposed into three sub-states. However, to understand the hybrid transition system of a model more clearly, this information can usually be hidden, just as has been done for the rest of the states in figure 2.

This example reveals two important points of labeled Pi calculus:

(1) Incremental Labels: The label *OPStatus* is unobservable initially for labeled process *ReceivePO*. It is observed and updated only when *InternalAct* is entered where the corresponding actions that can change the label *CurStatus* become 'free' in the process.

(2) Normalized Commitment: The construction of the labels in hybrid transition system is based on the normalized commitment defined in the previous section and can be outlined in the following steps (as is shown in the rectangles with dotted lines in figure 2):

1. Transition 'in' is fired and a new state is entered. The value of the labelset in the old state is thus updated according to the transition rules;

2. One of the expansion rules (*LE-RES*) is applied once and the labelset associated with a process composition '|' is identified as the necessary expansion;
3. Rules in table 1 are applied recursively to compute the labels contained in this expansion and the result is one new single label '*OPStatus*';
4. The new label '*OPStatus*' is then inserted into the existing labelset of the current state by the reduction function *LRed*. It is after this step that the whole labelset is completely determined associated with the new state.
5. Steps 1-4 are repeated such that the whole hybrid transition system is built.

We must stress here that not only existing Pi calculus related modal logics and verification algorithms can be used to check the process from a behavioral aspect in labeled Pi calculus, but also traditional temporal logic including CTL and LTL and other model checking methods can be exploited for the process verification from a state-oriented perspective which simply ignores the explicitly modeled actions. When both the action information and state information are needed in specification, there are also logics like Pi-logic[8], state/event derivative of LTL[14], etc that are ideal candidates for satisfying this purpose.

5. Hybrid Bisimulation and State Equivalency

Bisimulation analysis is an important tool for analyzing behavior equivalency between different processes in Pi calculus. Labeled Pi calculus inherits the core semantics of Pi calculus. For example, the reduction rules and structure congruence of Pi calculus remain unchanged. Besides, the extended transition rules are non-pervasive extension of the original transition rules. Therefore, existing bisimulation relations in Pi calculus still works for labeled Pi calculus. Their definitions are not explored in detailed again here. A comprehensive reference can be found in [2][12].

Considering the concept of labels and labelsets are newly introduced and the hybrid transition system does capture more information than labeled transition system does, this section is focus on how to integrate extra state information into existing bisimulation relations. Strong / weak open bisimulation [2] are used as a basis for the integration. Furthermore, the state simulation and equivalency relations are proposed to analyze system equivalency from pure state perspective as contrary to pure behavior equivalence defined in existing bisimulations.

Let $\Rightarrow^{\tau}$ denote some number of continuous transitions (possibly zero) fired by invisible action τ ; $\Rightarrow^{a}$ denote $\Rightarrow \xrightarrow{a} \Rightarrow$; $\Rightarrow^{\hat{a}}$ denote $\Rightarrow^{a}$ (or $\Rightarrow^{\tau}$) when $a \neq \tau$ (or $a = \tau$), and $\Rightarrow$ denote some number of continuous transitions (possibly zero) fired by any action, the bisimulations are defined as follows.

Definition 5.1 (Strong hybrid bisimulation): A symmetric binary relation *R* is a strong hybrid bisimulation if $(P, Ls^{P})R(Q, Ls^{Q})$ implies for any σ :

If $(P_{\sigma}, Ls^{P}) \xrightarrow{a} (P', Ls^{P}')$ where $bn(a) \notin fn(P_{\sigma}, Q_{\sigma})$, then there exists *Q'* such that $(Q_{\sigma}, Ls^{Q}) \xrightarrow{a} (Q', Ls^{Q}')$, $(P', Ls^{P}')R(Q', Ls^{Q}')$ and $Ls^{P}' = Ls^{Q}'$.

Definition 5.2 (Weak hybrid bisimulation): A symmetric binary relation *R* is a weak hybrid bisimulation if $(P, Ls^{P})R(Q, Ls^{Q})$ implies for any σ :

If $(P_\sigma, Ls^P) \xrightarrow{a} (P', Ls^P')$ where $bn(a) \notin fn(P_\sigma, Q_\sigma)$, then there exists Q' such that $(Q_\sigma, Ls^Q) \Rightarrow^{\hat{a}} (Q', Ls^Q')$, $(P', Ls^P')R(Q', Ls^Q')$ and $Ls^P' = Ls^Q'$.

Hybrid bisimulations are extensions of open bisimualtion, but the former is a more demanding relation since it also put state information into consideration. Equivalency relations which are defined from pure state perspective are discussed as below, which will be very useful for analyzing the relations between different process models.

Definition 5.3 (Strong State Simulation): A binary relation R is a strong state simulation if $(P, Ls^P)R(Q, Ls^Q)$ implies:

If $(P, Ls^P) \xrightarrow{a} (P', Ls^P')$, then there exists Q' and any action b, such that $(Q, Ls^Q) \xrightarrow{b} (Q', Ls^Q')$, $(P', Ls^P')R(Q', Ls^Q')$ and $Ls^P' \text{ p } Ls^Q'$.

Definition 5.4 (Strong State Equivalence): A binary relation R is a strong state equivalency relation if both R and R^- are strong state simulations, where R^- denotes the reverse relation of R.

Definition 5.5 (Weak State simulation): A binary relation R is a weak state simulation if $(P, Ls^P)R(Q, Ls^Q)$ implies:

If $(P, Ls^P) \xrightarrow{a} (P', Ls^P')$, then there exists Q' such that $(Q, Ls^Q) \Rightarrow (Q', Ls^Q')$, $(P', Ls^P')R(Q', Ls^Q')$ and $Ls^P' \text{ p } Ls^Q'$.

Definition 5.6 (Weak State Equivalence): A binary relation R is a weak state equivalency relation if both R and R^- are weak state simulations.

A similar state equivalency relation is also proposed in [9] which was regarded as a basis for optimizing business processes in their work. It defines a relatively relax relation in that only the initial and final states of processes are compared for qualifying two processes as state equivalent. However, the state simulations and equivalencies defined above can be viewed as a generalization for them. For example, by definition 5.6, the following conclusion can be got as similar in [9].

Lemma 5.1: Two labeled processes are *independent* if there is no shared transitions in their composition such that they can interact with each other by an internal action τ. Furthermore, denote *P{ls}* to represent a process *P* if *P* is explicitly associated with labelset *ls* in its process identifier. Then we have:

If *P{ls}* and *Q* are independent, then *P{ls}|Q* weakly simulates *P{ls};Q*, where ';' is a sequential operator defined in [1].

Proof: To prove the weak state simulation, notice that in Pi calculus the composition of two processes can be interpreted as an interleaving transition of each process. Therefore, there must be a transition sequence *TS* in *P{ls}|Q* that is identical with the one of *P{ls};Q*. Since the initial labelsets of two processes are all *ls*, it ensures that the labelsets of each state along *TS* are equivalent for both processes. Consequently a binary relation can be derived from *TS* such that it is a weak state simulation. □

Corollary 1: If *P{ls1}* and *Q{ls2}* are independent, then *P{ls1}|Q{ls2}* weakly simulates *P{ls1};Q{ls2}* if and only if *ls1=ls2*.

Proof: If *ls1=ls2* doest not hold, the initial labelsets of two processes are not the same although a similar transition sequence *TS* can still be found, i.e. (*LRed({}, ls1, ls2)* for *P{ls1}|Q{ls2}* and *LRed({}, ls1)* for *P{ls1};Q{ls2}*, where *{}* denotes an empty labelset. According to the expansion rules in table 2, this leaves the possibility that an

action associated in *ls1* can also change the labels in ls2 before the labelset *ls2* is expanded for *P{ls1};Q{ls2}*, which makes the states along *TS* to be nonequivalent. □

The reverse of Lemma 5.1 is obviously not hold, that is, *P{ls};Q* can not state simulate *P{ls}|Q* even if *P{ls}* and *Q* are independent.

The following lemmas are related to the interrelationships among state equivalency, hybrid bimimulation and open bisimulation.

Lemma 5.2: If *P* and *Q* are strong /weak hybrid bisimilar, it means that *P* and *Q* are both strong /weak open bisimilar and strong /weak label equivalent.

Proof: It directly follows from the definition of strong /weak hybrid bisimilation. □

It should be mentioned that the reverse of lemma 5.2 is not hold. That is, the fact that *P* and *Q* are both strong /weak open bisimilar and strong /weak label equivalent does not imply that *P* and *Q* are strong /weak hybrid bisimilar.

Taking the following processes as an example. *P* and *Q* are obviously strong open bisimilar and strong label equivalent, but they are not strong hybrid bisimilar.

$$Labelset\ lp =_{def} \underline{prop}^{\{0\}} : pola(a+, b-), Labelset\ lq =_{def} \underline{prop}^{\{0\}} : pola(a-, b+)$$

$$P(x,a,b)\{lp\} =_{def} (x.(a.0 + b.0))\{lp\} \qquad Q(x,a,b)\{lq\} =_{def} (x.(b.0 + a.0))\{lq\}$$

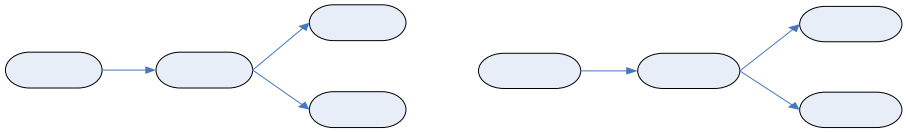

Fig. 3. A Non Strong Hybrid Bisimilar Example

Lemma 5.3: If *P* and *Q* are strong open bisimilar, it means that *P{ls}* and *Q{ls}* are strong label equivalent if reduction rule '*Hold*' and '*Supplement*' are never applied in reduction function *LRed* when new labeled are expaned during the transition of *P*, *Q*.

Proof: No application of rule '*Hold*' implies that no new labeled are inserted during the transition of *P*, *Q* while no application of rule '*Supplement*' implies that no association of new actions to existing labels are made during the transition of *P*, *Q*. Therefore, a similar proof can be made as in Lemma 5.1. First an identical action sequence *TS* can be found because of the strong open bisimulation, second the initial labelsets of *P* and *Q* and the label expansion on each state along *TS* are the same because rule '*Hold*' and '*Supplement*' are never applied. Thus a binary relation can be derived from *TS* such that it is a strong label equivalency. □

The above lemma obviously does not hold in the case of weak open bisimilar.

The benefits of label simulation and equivalency have three aspects:

- To compare two processes from a state perspective is a complement to the original behavioral observation of Pi calculus.
- Label simulation / equivalency and open bisimulation in Pi calculus are loosely coupled, label simulation / equivalency does not require new modal logics and algorithms for the analysis..
- The labels defined in each state reflect particular concerns about the process for different users. It is valuable for developing various reduction techniques for applying model checking to business process verification.

In fact, the third aspect is crucial for relieving state explosion problem in model checking. We will give some clues about several techniques that will be very useful

for maintaining the state space of labeled Pi calculus as below. Their implementation details are out the scope of this paper and will be discussed separately.

Label Reduction: Because there is a direct connection between actions and possible change of label values in hybrid transition system, only the changed labels are recorded instead of representing all the labels in a state. This can greatly decrease the state space.

Path Reduction: Because of the incremental nature of the labelsets illustrated in section 4, it is possible to find some transition paths where some labels never appear in them. This kind of paths can be regarded as redundant paths for those non-appeared labels. Some algorithms can be developed to eliminate these paths when checking a property that relates to the labels.

State Reduction: It is possible that some actions will never change the values of nay labels, i.e, some continuous states may be the same. Therefore, theses states can thus be abstracted into one state so as to reduce the size of the state space.

6. Conclusion

While applying Pi calculus to business process area, it is found that the Milner's l Pi calculus is not enough to capture the holistic semantics of business process model. An extension for Pi calculus, labeled Pi calculus, is proposed in this paper. The main idea of labeled Pi calculus is to integrate state information into the traditional Pi calculus. In labeled Pi calculus, a simple state name is replaced with the detailed state information. Two important concepts, label and labelset, are introduced in detail. As a result of this introduction, the labeled Pi calculus can also be viewed as a generalization of the fluent propositions in [18] where only polarity labels with '0-1' value are considered. In order to formally define the semantics of labeled Pi calculus, the extension of traditional labeled transition system, namely hybrid transition system, is discussed. Some properties of the extended semantics are also analyzed. From the semantics definition of labeled Pi calculus, it is clear that the extended Pi calculus is a simple process algebra which is able to integrate the concerned state information into the event-based process expressions. Just because of the loosely coupled mechanism, the semantics and theory of Pi calculus are well preserved in labeled Pi calculus. Some new state simulation and equivalency relations which can be regarded as the generalization of similar ideas in previous work [9] are introduced for the labeled Pi calculus. The relations between state equivalency, open bisimulation and hybrid bisimulation are also studied.

As one of its applications, labeled Pi calculus can be applied to formalize and analyze business process models. The models in labeled Pi calculus can capture both event-based information and state-based information, which is quite important and useful for understanding and analyzing business process models. On the other hand, such an extension also enables more choices of different process verification methods to be applied and new reduction techniques to be carried out in model checking.

Our future works include the application of the generalized state simulation and equivalency relations into the structural optimization purpose of processes as already been addressed in section 5. In addition, since this work mainly provides a modeling

framework from a theoretical perspective, tooling support is also one of the most important future works.

Reference

[1] Milner, R., *Communicating and Mobile Systems: the Pi-Calculus*. 1999, Cambridge: Cambridge University Press.

[2] Parrow, J., *An introduction to the Pi calculus*, in *Handbook of Process Algebra*. 2001, Elsevier Science.

[3] BPMI., *Business Process Modeling Language Specification* (Version 1.0), 2002, 11, http://www.bpmi.org

[4] Lumpe, M., *A Pi-Calculus Based Approach for Software Composition*[PhD Thesis], in Institut fur Informatik und angewandte Mathematik. 1999, der Universit at Bern.

[5] Pahl C., *A Pi-Calculus based Framework for the Composition and Replacement of Components.* in Proceeding of the Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA. 2001. p 97-107.

[6] W.M.P. van der Aalst. *The Application of Petri Nets to Workflow Management.* The Journal of Circuits, Systems and Computers, 1998. 8(1): p.21 - 66.

[7] Dam, M., *Model Checking Mobile Processes.* Information and Computation, 1996. 129(1): p. 35-51.

[8] Ferrari, G.-L., et al., *A Model-Checking Verification Environment for Mobile Processes.* ACM Transactions on Engineering and Methodology, 2003. 12(4): p. 440-473.

[9] Ying L., Ke X., et al., *BPEL4WS Semantics Formalization and Process Optimization with π-Calculus*. Submitted to the Fifth International Conference on Business Process Management.

[10] Ciobanu, G. and M. Rotaru, *A Pi-calculus Machine.* Journal of Universal Computer Science, 2000. 6(1): p. 39-59.

[11] Taguchi K., Dong JS., et al., *Relating Pi-calculus to Object-Z.* in IEEE International Conference on Engineering of Complex Computer Systems. 2004. p 97-106.

[12] Sangiorgi, D. and D. Walker, *The Pi-Calculus: a Theory of Mobile Processes*. 2001, Cambridge: Cambridge University Press.

[13] Lam, V. and J. Padget. *Formalization of UML statechart diagrams in the pi-calculus*. in Proceedings of the 13th Australian Software Engineering Conference. 2001. p 213-223.

[14] Chaki, S., et al. *State/Event-Based Software Model Checking*. in Proceeding of the 4th International Conference on Integrated Formal Methods. 2004. p 128-147.

[15] Object Management Group. Model Driven Architecture, http://www.omg.org/mda/, 2004

[16] Grieskamp W., Santen T., Stoddart B. editors. IFM' 00: Integrated Formal Methods. Lecture Notes in Computer Science volume 1945. 2000.

[17] Butler M., Petre L., K. Sere. editors. IFM' 02: Integrated Formal Methods. Lecture Notes in Computer Science volume 2335. 2002.

[18] Giannakopoulou D., Magee J. Fluent model checking for event-based systems. In Proceedings of FSE. 2003. p. 257-266

[19] Kindler E., Vesper T. ESTL: A temporal logic for events and states. In Proceedings of ATPN 98. 1998. p. 365 - 383