A New-type Pi Calculus with Buffers and Its Bisimulation

Hui Kang
College of Computer Science and Technology, Jilin University, Changchun, Jilin Province, China
Email: kanghui@jlu.edu.cn

Zhi Wang, Shuangshuang Zhang and Fang Mei
College of Computer Science and Technology, Jilin University, Changchun, Jilin Province, China
Email: jlu20254@163.com, zss198906@163.com, meifang@jlu.edu.cn

***Abstract*—According to the features of asynchronous interaction in systems, a new-type Pi calculus with buffers — Buffer-Pi calculus is proposed, the new labelled transition system based on buffers is introduced, the enhanced describing capability is shown to apply Buffer-Pi calculus to modeling with the concrete example of asynchronous interaction, and the new behavior equivalence relations are defined, several propositions and properties of Buffer-Pi calculus are given. The study shows that compared with Pi calculus, Buffer-Pi calculus can provide more powerful support for asynchronous behavior modeling in systems.**

***Index Terms*—asynchronous interaction, Buffer-Pi calculus, the labelled transition system with buffers, behavior equivalence relation**

I. INTRODUCTION

In the past few decades, with the development of computer science and communications technology, interactive systems have become increasingly popular, and the way of computing has changed a lot from the traditional sequential calculation [1][2] extended to the concurrent computing with interaction [3][4] as the main feature, simultaneously both location [5-8] and mobility [9][10] are characterized as independent concepts in the formal model. Pi calculus [11] proposed by Robin Milner is a kind of process algebra method describing concurrent interactive systems, whose basic entities are processes and names. The syntax and semantics of Pi calculus can be found in the Reference [11-14], and they are not repeated here. Pi calculus is a computational model based on synchronization interactive pattern with a lack of asynchronous features. However, with the development of network technology, asynchronous communication technology with Ajax [15][16] as the representative has become a mainstream, so it requires a corresponding formal method to describe and analyze systems with asynchronous features [17].

In previous work, there are two main methods to handle asynchronous interaction in the process algebra framework represented by Pi calculus: One method is to use the parallel operator instead of output prefix [18], for example, $X = \bar{a}(m_1).\bar{b}(m_2).\bar{c}(m_3).P$ represents sending the messages m1, m2, m3 in turn through the channels a, b and c, and then executing P (Here P means the remaining calculation of the process X). The above formula only reflects the sequential features of sending messages, but not asynchronous features of system interaction. Then the above formula can be changed into $X = \bar{a}(m_1) \mid \bar{b}(m_2) \mid \bar{c}(m_3) \mid P$, and in this formula asynchronous features are shown, while the messages cannot reach the other side of the interaction in turn. The other method is to send a message by constructing components such as a buffer [19]. In this case, the above formula can be shown as $X = \bar{i}'(m).P$, which indicates that the buffer i' receives the output message m, and at the same time P can be executed without waiting. These two methods both have advantages and disadvantages: The first method is simple, but the messages may reach without order; the second method is practical, but it is too complicated.

Taking the above into account, a new-type Pi calculus with buffers is proposed, called Buffer-Pi calculus in this paper. Buffer-Pi calculus brings the abstract concept of asynchronous into Pi calculus through the concrete form of buffer components, which makes the interactive way of synchronous and asynchronous in the system be described. This is different from that shown in Reference [19]. In this paper, the buffer is presented as an independent concept, and achieves the purposes of coordinating and controlling asynchronous behavior by interacting with the Scheduler (detailed description to be given below). It combines the advantages of two methods mentioned above—parallel and buffer mechanisms to enhance the describing capability of asynchronous interaction of Pi calculus. By associating the new buffer operators with the action of Pi calculus dynamically, and merging the semantics of the buffer operators with the operational semantics of Pi calculus together, the integration and interaction of buffers and basic Pi calculus are achieved, and extending basic Pi calculus directly is avoided, thus it reduces the complexity and makes up the shortage that basic Pi calculus cannot construct the model of asynchronous interaction systems directly. In this

doi:10.4304/jcp.7.12.2931-2938

paper, The main work is as follows: (1) proposing a new-type of Pi calculus with buffers—Buffer-Pi calculus; (2) in the concept of buffers, putting forward a new labelled transition system with buffers; (3) giving the new behavioral equivalence of Buffer-Pi calculus on the basis of (2); (4) the propositions and properties of Buffer-Pi calculus are given using the definition in (3).

II. The Syntax of Buffer-Pi Calculus, the Labelled Transition System with Buffers, and Extended Operational Semantics

A. Basic Concepts

Before the syntax of Buffer-Pi calculus is given formally, the following basic concepts are offered at first.

Definition 1 Basic Type and Compound Type: As mentioned above, there are two kinds of basic entities in Pi calculus—processes and names. Names can represent channels, and one new channel can be transmitted by a known channel. Processes received can use the new channel to establish a connection, so the type of CHANNEL is introduced for these names. Names can also represent data information, such as integer, boolean string and so on, therefore the type of VALUE is introduced for them. Therefore, the basic type $\hat{t}$ can be defined as follows:

$\hat{t} ::= CHANNEL | VALUE$

Besides the basis type, the operators of type structure can be used to construct the compound type. For instance, Cartesian production '$\times$' can be used to construct the type of dual. With the level structure of names introduced here, $LAYER$ is defined as the operator of level structure. The type hierarchy consisting of $LAYER$ and sequence of all types can be defined now, such as:

$LAYER(), LAYER(LAYER(LAYER())), LAYER(()), \cdots$

Definition 2 Buffer Cell: A buffer cell is a triple, Cell=(ident, type, port), where ident is the unique identifier of a buffer cell(the ident of cells in the buffer is buffer identifier and location index), the type is the entity type stored in buffer cells, and the port is used for buffer cells to interact with the outside, just as Fig. 1 shows:

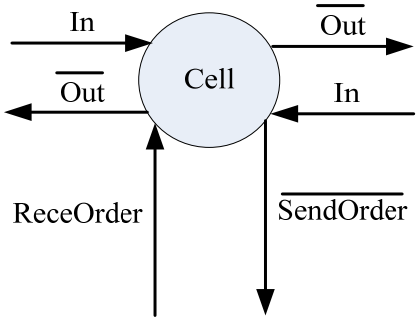

Figure 1. The basic structure of a Cell

It can be seen from Fig. 1 that the Cell can complete the two-way transmission of information (in In and $\overline{Out}$ ports). As a result of the symmetry of information transmission, this paper only makes formal description and analysis of a single direction of information transmission without special instruction.

In order to facilitate the formal description of the Cell and system modeling below, some notations will be introduced. $A(p_1,\cdots,p_n)$ denotes a definition identifier of the process, and $(p_1,\cdots,p_n)$ is its port parameter list. $port<c_1,\cdots,c_n>$ indicates that processes can send or receive commands by port, and $<c_1,\cdots,c_n>$ is its command parameter list. $A(p_1,\cdots,p_n)_{\rightarrow} port<c_1,\cdots,c_n>$ is the port action with its command parameter list when reaching the process A.

As for the Cell, there are the following ports:

In : receiving outside information, and storing it in the Cell.

$\overline{Out}$: sending the information of the Cell to the outside.

$\mathrm{Re}ceOrder$: receiving commands from the Scheduler. When the Cell receives the command "RedayReceive" from the port $\mathrm{Re}ceOrder$, the port In will start to receive data; when receiving "ReadySend", the port $\overline{Out}$ will start to send data.

$\overline{SendOrder}$: the Cell sending commands to the Scheduler. When there is content stored in the Cell, the command "CanSendData" will be sent, and the Scheduler will be told that the Cell is in the state that can sent data and will block the port In to receive data. When there is nothing in the Cell, the command "CanReceiveData" will be sent, and the Scheduler will be told the Cell is in the state that can receive data and will block the port $\overline{Out}$ to send data.

The definition of $Cell(In,\overline{Out},\mathrm{Re}ceOrder,\overline{SendOrder})$ is given, and:

$$\begin{aligned} Cell &\overset{def}{=} \mathrm{Re}ceOrder<\mathrm{Re}ady\,\mathrm{Re}ceive>. \\ &\quad In.Cell'_{\mapsto}\overline{SendOrder}<CanSendData> \end{aligned} \tag{1}$$

$$\begin{aligned} Cell' &\overset{def}{=} \mathrm{Re}ceOrder<\mathrm{Re}adySend>. \\ &\quad \overline{Out}.Cell_{\mapsto}\overline{SendOrder}<Can\,\mathrm{Re}ceiveData> \end{aligned}$$

From the perspective of process algebra, the definition of the Cell is a process expression with certain action.

Definition 3 Buffer: It is an ordered set composed of the same type of buffer cells. A buffer is a triple. Buffer = (ident, type, n), where the ident is the unique identifier of buffer, the type means data type stored in buffer cells, and n indicates the current buffer size, whose structure can be expressed as follows:

$$(type)ident^{(n)} ::= \{Cell_1^{(n)},\cdots,Cell_v^{(n)},\cdots,Cell_n^{(n)}\} \tag{2}$$

The current buffer size n is also known as the length of the buffer, and it can be obtained by $length(ident)$, $\max(ident)$ means the maximum capability that the buffer can apply, and $type(ident)$ means the data type of the buffer. Here is an example of defining a buffer below. $(CHANNEL)Buff^{(n)}$ means that the data type of the buffer named Buff is CHANNEL, whose current buffer size is n, and the current buffer size n is dynamic

with the increase and decrease of buffer cells. $Cell_v^{(n)}$ means the vth Cell in the buffer with the size n.

The component Buffer in this paper is used in asynchronous interaction of the system. As stated in the introduction, sending or receiving asynchronous messages is directional and sequential, so the Buffer in the definition is an ordered set, and the first Cell ($Cell_1^{(n)}$) in the Buffer is called the Header, the last Cell ($Cell_n^{(n)}$) is called the Tail.

In terms of the entity type stored in buffers, the buffer can be divided into the channel buffer (ChannelBuffer), the data buffer (ValueBuffer) and the composite buffer (CompoundBuffer). In addition, processes during the execution can create a buffer with several processes, and the type of interactive information may be CHANNEL or DATA, even there may be more complex compound type, so the set consisting of all kinds of buffers for the specific process is named buffer set, denoted as BufferSet.

In order to distinguish synchronous or asynchronous interaction of the system, the port action between the processes is abstracted into two classes, one is the asynchronous port action, denoted as [port], and the other is the synchronous port action, denoted as <port>(Buffer-Pi calculus syntax will be described further in the following).

Definition 4 Buffer Chain: Intuitively, it links the right port ($\overline{Out}$)of a cell to the left port(In) of the next cell, and adjacent cells are linked by port just as Fig. 2 shows. The below is the formal definition of a binary chain and brings in the link operator $\frown$:

$$Cell_v \frown Cell_{v+1} \overset{def}{=} new l(\{l / \overline{Out}\}Cell_v \mid \{l / In\}Cell_{v+1}) \quad (3)$$

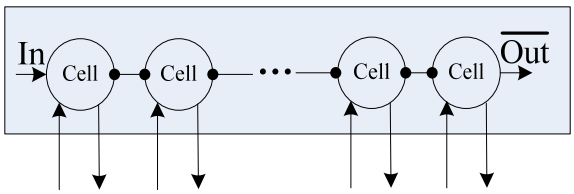

Figure 2. Weak equivalence chain structure of the Buffer

Fig. 2 shows the weak equivalence chain structure of the Buffer. The ports $\mathrm{Re}\,ceOrder$ and $\overline{SendOrder}$ are omitted and only a single direction of information transmission is given, but this does not affect the external function of the Buffer.

Since the topic of this paper is the component Buffer, so here the functional description of the Scheduler is only given, as shown in Fig. 3, where a_v is the control interaction with the port $\mathrm{Re}ceOrder$ of $Cell_v^{(n)}$, and b_v is the control interaction with the port $\overline{SendOrder}$ of $Cell_v^{(n)}$. By means of the command interaction of the Schedule and the Buffer, it ensures that the external action of the Buffer keeps coordination and order.

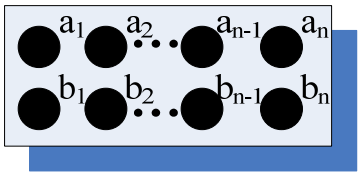

Figure 3. The structure of the Scheduler

Proposition 1 $Buff^{(n)} \approx \overbrace{Cell \frown \cdots \frown Cell}^{n}$ (4)

Proof: In the Reference [20] the similar property is given: it indicates that the buffer $Buff^{(n)}$ independently composed of n cells is weak equivalent to the components linked by n cells. However, the cell structure proposed in this paper is different from that in the Reference [20]. The buffer cell in the Reference [20] is a simple process structure only with left and right port while the cell structure is refined in this paper. As the cell structure in this paper is extended on the basis of that in the Reference [20], the property above can also be applied in this paper.

□

Moreover, in Fig. 2, the function ports of the Buffer do not change much compared with that in the Cell, and the information ports interacting with the outside does not increase (In and $\overline{Out}$). Only the control ports interacting with the Scheduler is changing in the quantity with the variation of the buffer size ($\mathrm{Re}ceOrder$ and $\overline{SendOrder}$). Therefore the formal definition of the Buffer has no essential difference with the Cell, and it is omitted. $Buff\_Controller \overset{def}{=} new(\mathrm{Re}ceOrder, \overline{SendOrder})(Buff \mid Scheduler)$ will be used as a whole in order to facilitate modeling.

The complementary actions of In and $\overline{Out}$ are defined as follows.

In order to interact with the $Buff\_Controller$, the complementary actions of In and $\overline{Out}$ are defined as follows:

$\overline{In}(\overrightarrow{name})$: It indicates the interactive mode of asynchronous output from processes to buffers. The parameter $\overrightarrow{name}$ means the information stored in the buffer successively according to the order of the process output. By executing in parallel with the process, it can interact with the Buff_Controller. Before asynchronous output in the process, the buffer must enter the state which can receive data, and can be operated through the buffer expression associated with the evolutionary action. When processes have finished asynchronous output, new information will be added to the tail of the parameter $\overrightarrow{name}$.

$Out(\overrightarrow{name})$: It indicates the interactive mode of asynchronous input of processes from the information of the buffers. The parameter $\overrightarrow{name}$ means the information stored in the buffer successively according to the order of the process's input. By executing in parallel with the process, it can interact with Buff_Controller. Before asynchronous input in the process, the buffer must enter the state which can send data, and can be operated

through the buffer expression associated with the evolutionary action. When processes have finished asynchronous input, information which has been send will be deleted from the header of the parameter $\overrightarrow{name}$.

Definition 5 Buffer Partial Order and Buffer Equivalency: For the Buffer B_1 and the Buffer B_2, they are denoted as $B_1 \prec B_2$, if $type(B_1) = type(B_2)$, and $length(B_1) \le length(B_2)$.

For the Buffer B_1 and the Buffer B_2, they are denoted as $B_1 = B_2$, if $B_1 \prec B_2$ and $B_2 \prec B_1$.

Definition 6 BufferSet Partial Order and BufferSet Equivalency: For the two BufferSets $BufferSet_1$ and $BufferSet_2$ owned by the processes P_1 and P_2, they are denoted as $BufferSet_1 \lhd BufferSet_2$, if $|BufferSet_1| = |BufferSet_2|$ and to each element B_1 in $BufferSet_1$, there is a unique element B_2 corresponding with, making $B_1 \prec B_2$ be true, and the contrary is also true.

For the two BufferSets $BufferSet_1$ and $BufferSet_2$ owned by the processes P_1 and P_2, they are denoted as $BufferSet_1 \hat{=} BufferSet_2$, if $|BufferSet_1| = |BufferSet_2|$ and to each element B_1 in $BufferSet_1$, there is a unique element B_2 corresponding with, making $B_1 = B_2$ be true, and the contrary is also true.

B. The Syntax of Buffer-Pi Calculus

For Buffer-Pi calculus, a crucial task is to determine how the system's dynamic execution semantics be associated with the Buffer's creation, destruction and update, which comes true by Buffer's operators, and Fig. 4 shows the complete syntax of Buffer-Pi calculus.

$$
\begin{aligned}
P &::= [\pi]\{BufferExp\}.P \mid \pi.P \mid P \mid Q \\
&\quad \mid P + Q \mid !P \mid new\ a\ P \\
&\quad \mid [C]P_1 : P_2 \mid A(x_1, x_2, \cdots, x_n) \mid 0 \\
[\pi]\{BufferExp\} &::= [x](\vec{y})\{BufferExp\} \\
&\quad \mid [\bar{x}](\vec{y})\{BufferExp\} \\
&\quad \mid [\tau]\{BufferExp\} \mid NUL(BufferExp) \\
<\pi>.P &::= <x>(\vec{y}) \mid <\bar{x}>(\vec{y}) \mid \tau \\
C &::= [x = y] \mid BoolExp \mid C \vee C \mid C \wedge C \mid \neg C \\
BufferExp &::= (BufferOp, B) \\
&\quad \mid BufferExp, BufferExp \\
&\quad \mid \Phi \quad (\Phi\ denates\ the\ null) \\
BufferOp &::= \Delta \mid \nabla \mid + \mid - \\
Buffer &::= (ident, type, n)
\end{aligned}
$$

Figure 4. The syntax of Buffer-Pi calculus

In Fig. 4, two prefix actions are defined, one is asynchronous interaction associated with the buffer expression($[\pi]\{BufferExp\}$), and the other is synchronous interaction ($<\pi>.P$). $\pi\{BufferExp\}$ introduces a special action $NUL(BufferExp)$, which means to operate the buffer independently without the evolutive action of the system. Buffer operation implements a relation of buffer conversion $\Re : BufferSe\ t' \times BufferOp \times B \to BufferSe\ t''$, that is to say, the buffer set which exists in the current system and the buffer expression can identify a new buffer set. In particular, there are four kinds of buffer operation in Buffer-Pi calculus, they are as follows:

1 creation of buffers (Δ): create a new buffer, specifying the buffer type and the initial length n.

2 destruction of buffers (∇): remove the specified buffer from the BufferSet of the current process.

3 addition of buffer cells (+): add a new buffer cell to the Tail of a buffer.

4 deletion of buffer cells (-): delete the buffer cell from the Header of a buffer.

In order to facilitate the unified representation below, $[\alpha]$ signify the asynchronous action in the system associated with the buffer expression, that is to say $[\alpha] = [\pi]\{BufferExp\}$, $<\alpha>$ signifies synchronous action in the system, and α signify the abstract action in the general sense. The key point of this paper is the properties of systems in asynchronous interaction, and synchronous action in the system is the same as basic Pi calculus, therefore in the case of no special instruction, the following discussions all aim at the asynchronous interaction.

C. The Labelled Transition System with Buffers

Based on the above definition, a new labelled transition system is presented in this section—the labelled transition system with buffers.

Definition 7 the Labelled Transition System with Buffers: In the Buffer-Pi calculus, the labelled transition system with buffers $(S, M, \{\xrightarrow{[\pi]\{BufferExp\}} \mid \pi \in M\})$ consists of the following elements: S is a two-tuple set consisting of processes and the BufferSet associated with it, M is the labelled set, while the transition set is $\{\xrightarrow{\{BufferExp\}[\pi]}\} \subseteq S \times S, \pi \in M$.

In the labelled transition system with buffers, the transition $\{BufferSe\ t'\} :: P' \xrightarrow{[\pi]\{BufferExp\}} \{BufferSe\ t''\} :: P''$ indicates that the current process P' has the buffer set named $BufferSet'$, and through implementing an asynchronous action $[\pi]$ which is associated with the buffer expression $BufferExp$, P' is converted into P'' with a buffer set named $BufferSe\ t''$. In order to conveniently make a description of the Buffer-Pi calculus asynchronous operational semantics, the method Trans is defined strictly as follows:

$$
\begin{aligned}
&Trans : BufferSe\ t' \times BufferExp \to BufferSe\ t'' \\
&Trans(BufferSe\ t', BufferExp) \\
&= \begin{cases} \Re(BuffSet', BufferOp, B) \\ if\ \ BuferfExp = (BufferOp, B) \\ BufferSe\ t' \\ if\ \ BufferExp\ \ is\ null \end{cases}
\end{aligned}
$$

$$Out: \frac{}{\varphi :: \bar{a}(b)\{\delta\}.P \xrightarrow{[\bar{a}](b)\{\delta\}} Trans\ (\varphi,\delta) :: P} \qquad Inp: \frac{}{\varphi :: a(b)\{\delta\}.P \xrightarrow{[a](c)\{\delta\}} Trans\ (\varphi,\delta) :: P\{c/b\}}$$

$$Tau: \frac{}{\varphi :: \tau\{\delta\}.P \xrightarrow{[\tau]\{\delta\}} Trans\ (\varphi,\delta) :: P} \qquad Buffer: \frac{}{\varphi :: P \xrightarrow{NUL\{\delta\}} Trans\ (\varphi,\delta) :: P}$$

$$Sum: \frac{\varphi :: P \xrightarrow{[\alpha]} \varphi' :: P'}{\varphi :: P+Q \xrightarrow{[\alpha]} \varphi' :: P'+Q} \qquad Par: \frac{\varphi :: P \xrightarrow{[\alpha]} \varphi' :: P'}{\varphi :: P \mid Q \xrightarrow{[\alpha]} \varphi' :: P' \mid Q}$$

$$\text{Res}: \frac{\varphi :: P \xrightarrow{[\alpha]} \varphi' :: P'}{\varphi :: (newv)P \xrightarrow{[\alpha]} \varphi' :: (newv)\,P'} \quad v \neq n(\alpha) \qquad \text{Rep}: \frac{\varphi :: P \xrightarrow{[\alpha]} \varphi' :: P'}{\varphi :: !P \xrightarrow{[\alpha]} \varphi' :: P' \mid !P}$$

$$Open: \frac{\varphi :: P \xrightarrow{[\bar{x}](z)\{\delta\}} Trans\ (\varphi,\delta) :: P'}{\varphi :: (newz)P \xrightarrow{[\bar{x}](z)\{\delta\}} Trans\ (\varphi,\delta) :: P'} \quad z \neq x$$

$$Match: \frac{\varphi :: P \xrightarrow{[\alpha]} \varphi' :: P'}{\varphi :: [C]P \xrightarrow{[\alpha]} \varphi' :: P'} \quad C \text{ means that the condition is true}$$

$$Com: \frac{\varphi :: P \xrightarrow{[\bar{a}](b)\{\delta\}} Trans\ (\varphi,\delta) :: P', \varphi' :: Q \xrightarrow{[a](c)\{\delta'\}} Trans\ (\varphi',\delta') :: Q'}{\varphi :: P \mid \varphi' :: Q \xrightarrow{[\tau]\{\delta,\delta'\}} Trans\ (\varphi,\delta) :: P' \mid Trans\ (\varphi',\delta') :: Q'}$$

$$Close: \frac{\varphi :: P \xrightarrow{[\bar{a}](b)\{\delta\}} Trans\ (\varphi,\delta) :: P', \varphi' :: Q \xrightarrow{[a](c)\{\delta'\}} Trans\ (\varphi',\delta') :: Q'}{\varphi :: P \mid \varphi' :: Q \xrightarrow{[\tau]\{\delta,\delta'\}} newz\ (Trans\ (\varphi,\delta) :: P' \mid Trans\ (\varphi',\delta') :: Q')} \quad z \notin fn(Q)$$

Figure 5. The asynchronous operational semantics of Buffer-Pi calculus

D. The Extended Operational Semantics of Buffer-Pi calculus

Fig. 5 gives the asynchronous operational semantics of Buffer-Pi calculus, where φ and δ respectively means BufferSet and BufferExp, φ' is the buffer set after the buffer operation associated with asynchronous action $[\alpha]$, and fn and bn respectively means the free name set and the bound name set.

In Fig. 5, the asynchronous operational semantics of Buffer-Pi calculus is given, and the synchronous operational semantics is the same as that of basic Pi calculus, which can be found in the Reference [11], here it isn't repeated. It should be pointed out that the asynchronous operational semantics of Buffer-Pi calculus can not only do buffer operation through the buffer expression associated with process action, such as Out, Inp and Tau, but also can do the direct operation to buffers through $NUL(BufferExp)$.

E. The Modeling and Analysis of Processes with Asynchronous Interaction based on Buffer-Pi Calculus

As shown in Fig. 6, a simple example of processes with asynchronous interaction is given. The asynchronous messages m_1, m_2 are passed on through the port α, and the synchronous message m_3 is passed on through the port β. There is only one buffer with the type CHANNEL in the BufferSet between P and Q, but it explains the problem fully.

First of all, according to the basic Pi calculus, the formal description of the whole system can be made as follows:

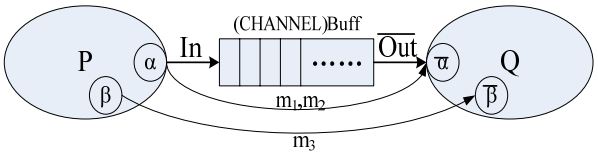

Figure 6. Processes with asynchronous interaction based on Buffer

$$P = \bar{\alpha}(m_1).\bar{\alpha}(m_2).\bar{\beta}(m_3).P'$$

$$Q = \alpha(m_1').\alpha(m_2').\beta(m_3').Q'$$

$$System = P \mid Q$$

After introducing the component Buffer, the system can be modeled by using Buffer-Pi calculus:

$$(CHANNEL)Buff^2 = \{m_1, m_2\}$$

$$Buff :: P = \overline{In}(m_1, m_2) \mid Buff :: < \bar{\beta} > (m_3).P'$$

$$Buff :: Q = out(m_1, m_2) \mid Buff :: < \beta > (m_3).Q'$$

$$Buff\_Controller = new(\text{Re}ceOrder, \overline{SendOrder})\ (Buff \mid Scheduler)$$

$$Buff :: System = Buff :: (Buff\_Controller \mid P \mid Q)$$

For the Buff in the above definition, when the process P needs to send the new information m_4 to Q, the transition of basic Pi calculus is $P' \xrightarrow{\bar{\alpha}(m_4)} P''$, and the following transition can be gained through applying Buffer-Pi calculus:

$$Buff :: System \xrightarrow{[\overline{\alpha}](m_4)\{(Buff,+)\}} Buff' :: (\overline{In}(m_1,m_2,m_4) \mid < \overline{\beta} > (m_3).P'' \mid Q \mid Buff\_Controller')$$

$$Buff\_Controller' = new(\mathrm{Re}ceOrder, \overline{SendOrder})(Buff' \mid Scheduler)$$

$$(CHANNEL)Buff'^3 = \{m_1, m_2, m_4\}$$

For the Buff, when the process Q receives the asynchronous message m_1, the transition of the basic Pi calculus is $Q \xrightarrow{\alpha(m_1')} \alpha(m_2).\alpha(m_3).Q'$, the following transition can be gained through applying Buffer-Pi calculus:

$$Buff :: System \xrightarrow{[\alpha](m_1')\{(Buff,-)\}} Buff' :: (\overline{Out}(m_2) \mid < \beta > (m_3).Q' \mid P \mid Buff\_Controller')$$

$$Buff\_Controller' = new(\mathrm{Re}ceOrder, \overline{SendOrder})(Buff' \mid Scheduler)$$

$$(CHANNEL)Buff'^1 = \{m_2\}$$

III. Buffer Bisimulation and Its Property Analysis

Bisimulation equivalence is the core issue of process algebra. In basic Pi calculus, strong / weak bisimulation equivalence have been defined [11], while in Buffer-Pi calculus, the impact which buffer operations bring in need to be considered, since the component Buffer is introduced. So in this section the concept of bisimulation will be redefined on the basis of the labelled transition system with buffers defined above. To be convenient, the following symbols are given at first: $\Rightarrow$ indicates a transition sequence composed of a number of actions that are not visible (its length can be 0); $\overset{\alpha}{\Rightarrow}$ means $\Rightarrow\overset{\alpha}{\rightarrow}\Rightarrow$, where α is a visible action. $\overset{\hat{\alpha}}{\Rightarrow}$ means if $\alpha = \tau$ then $\overset{\hat{\alpha}}{\Rightarrow} = \Rightarrow$, and if $\alpha \neq \tau$, then $\overset{\hat{\alpha}}{\Rightarrow} = \overset{\alpha}{\Rightarrow}$; $\overset{\alpha}{\Rightarrow}$ means an action sequence with any length. A simplified representation of processes in Buffer-Pi calculus is given below. For a process expression of basic Pi calculus P, using the asynchronous action prefix $[\alpha]$ and the synchronous action prefix $<\alpha>$, the action of asynchronous and synchronous in the process P is distinguished. For example, $P = \overline{\alpha}(m_1).\overline{\alpha}(m_2).\overline{\beta}(m_3).P'$ is changed into $P = [\overline{\alpha}](m_1).[\overline{\alpha}](m_2). < \overline{\beta} > (m_3).P'$.

Definition 8 Strong Mixed Buffer Bisimulation: the symmetric binary relation R in Buffer-Pi calculus is called strong mixed buffer bisimulation, if to $(BufferSet_P :: P)R(BufferSet_Q :: Q)$, $BufferSet_P :: P \xrightarrow{\alpha} BufferSet_P' :: P'$, then there exists a Q', which makes $\max(BufferSet_P') = \max(BufferSet_Q')$, $BufferSet_P' \hat{=} BufferSet_Q'$, and $(BufferSet_P' :: P')R(BufferSet_Q' :: Q')$.

Definition 9 Weak Mixed Buffer Bisimulation: the symmetric binary relation R in Buffer-Pi calculus is called weak mixed buffer bisimulation, if to $(BufferSet_P :: P)R(BufferSet_Q :: Q)$, $BufferSet_P :: P \xrightarrow{\alpha} BufferSet_P' :: P'$, then there exists a Q', which makes $\max(BufferSet_P') = \max(BufferSet_Q')$, $BufferSet_P' \hat{=} BufferSet_Q'$, and $(BufferSet_P' :: P')R(BufferSet_Q' :: Q')$.

In order to observe the impact of the buffers' introduction separately, the buffer is considered as an independent dimension of system description, and the system equivalence is described from the perspective of the component Buffer. In fact, abstracting out the buffer to make equivalence description can clearly describe the system's asynchronous interactivity, which contains the number and type of asynchronous interaction. Therefore it is quite necessary to do research on the asynchronous properties of interactive systems by defining the equivalence of buffers independently.

Definition 10 Buffer Simulation and Buffer Bisimulation: the symmetric binary relation R in Buffer-Pi calculus is called buffer simulation, if to $(BufferSet_P :: P)R(BufferSet_Q :: Q)$, $BufferSet_P :: P \xrightarrow{\alpha} BufferSet_P' :: P'$, then there exists a Q', which makes $\max(BufferSet_P') = \max(BufferSet_Q')$, $BufferSet_P' \hat{=} BufferSet_Q'$, and $(BufferSet_P' :: P')R(BufferSet_Q' :: Q')$.

The symmetric binary relation R in Buffer-Pi calculus is called buffer bisimulation, only if R and its inverse R^{-1} both are buffer simulation.

Proposition 2 Buffer bisimulation is a kind of equivalence relation.

Proof: The reflexivity is obviously true. To prove the symmetry, it needs to prove that if the relation R is buffer bisimulation, then its inverse R^{-1} must be buffer bisimulation. From the definition of buffer bisimulation, it is obvious; For the transitivity, it must prove that if R_1 and R_2 both are buffer bisimulation, then their composite relation is $R_1R_2 = \{(BufferSet_p :: p, BufferSet_r :: r) \mid \exists q \ \ pR_1q \ and \ qR_2r\}$ is also buffer bisimulation. It only needs to be enough to verify that R_1R_2 is buffer simulation. Suppose $(BufferSet_p :: p, BufferSet_r :: r) \in R_1R_2$ and $BufferSet_p :: p \xrightarrow{\alpha} BufferSet_{p'} :: p'$. Because there exists q to meet $(BufferSet_p :: p)R_1(BufferSet_q :: q)$ and $(BufferSet_q :: q)R_2(BufferSet_r :: r)$, then also exists q', which makes $BufferSet_q :: q \overset{\mu}{\Rightarrow} BufferSet_{q'} :: q'$, $\max(BufferSet_p') = \max(BufferSet_q')$, $BufferSet_p' \hat{=} BufferSet_q'$ and $(BufferSet_p' :: p')R(BufferSet_q' :: q')$, so there also exists an r', which makes $BufferSet_r :: r \overset{\mu}{\Rightarrow} BufferSet_{r'} :: r'$, $\max(BufferSet_q')$

$= \max(BufferSet_r')$, $BufferSet_q' \triangleq BufferSet_r'$ and $(BufferSet_q' :: q')R_2(BufferSet_r' :: r')$ and, so $(BufferSet_p' :: p', BufferSet_r' :: r') \in R_1R_2$, therefore it verify that R_1R_2 is buffer simulation. □

Proposition 3 If the process P and Q with buffers are strong mixed buffer bisimulation or weak mixed buffer bisimulation, and then they must be buffer bisimulation.

Proof: From the definition of mixed buffer bisimulation and buffer bisimulation it can be obtained directly.

Proposition 3 has proved that the buffer bisimulation is a looser equivalence relation relative to strong/weak mixed buffer bisimulation, which reflects the simulation of buffer state equivalent during the process evolution.

If there are processes P, Q and R, and there is a buffer $BufferP$ between P and R, and a buffer $BufferQ$ between Q and R, when $type(BufferP) = type(BufferQ)$ and $\max(BufferP) = \max(BufferQ)$, it can be said that P and Q have the same asynchronous processing capability of the same type, which is denoted as $P \underset{R}{\Diamond} Q$. When $type(BufferP) = type(BufferQ)$ and $\max(BufferP) > \max(BufferQ)$, it shows that P has a stronger asynchronous processing capability than Q, which is denoted as $P \underset{R}{\overline{\triangleright}} Q$.

Property 1 If $P \underset{R}{\Diamond} Q$ and $type(BufferP) = type(BufferQ) = CHANNEL$, then P and Q have the same asynchronous structure converting capability.

Property 2 If $P \underset{R}{\Diamond} Q$ and $type(BufferP) = type(BufferQ) = DATA$, then P and Q have the same asynchronous basic data transferring capability.

Property 1 and Property 2 give a standard to judge asynchronous processing capability of the specified processes, which makes the processes' asynchronous processing capability quantified and typed.

IV. SUMMARY

A. Analysis of Asynchronous Processing Capability

In fact, in the Reference [18][19], the processing approaches of asynchronous behavior are simply given, and their properties are not systematically discussed. Property 1 and Property 2 show the quantified standard of asynchronous processing capability in the system. For example, the process P with asynchronous behavior, the process expression in the form of Pi calculus is defined as follows:

$$P = \overline{b}(x) \mid b(y).\overline{a}(y).\overline{y}(u) \mid a(z).\overline{z}(v)$$

where x is a name with the type CHANNEL of asynchronous features, and other free names is of synchronous features. Applying the methods of the Reference [18][19], asynchronous behavior of the process P isn't presented, but using the Buffer-Pi calculus, it can be expressed as follows:

$$P = \overline{b}(< x >) \mid b(y).\overline{a}(y).\overline{y}(u) \mid a(z).\overline{z}(v)$$

It can be seen that there are no subject names to ports of asynchronous features in the above formula. Therefore, for the current process P, it doesn't need the component buffer. When the process P proceeds the further action, it is changed into $p' = < \overline{x} > (u) \mid < \overline{x} > (v)$, where there is the asynchronous port x on two subject positions. It illustrates at least the process P needs a buffer of $\max(Buffer_P) = 2$ in order to take full advantage of the asynchronous processing capability of the process P.

B. Conclusion and Future Research

In this paper, according to the asynchronous features of interaction systems, a new-type Pi calculus with buffers—Buffer-Pi calculus is proposed on the basis of basic Pi calculus, the labelled transition system with buffers is defined, the Buffer-Pi calculus is applied to make a formal description of specific asynchronous processes with asynchronous interaction, and it is shown that the extended Buffer-Pi strengthens the capability to describe system asynchronous behavior, and then the new behavior equivalence relations are proposed between the processes, and in the end several propositions and properties of the Buffer-Pi calculus are given.

On the basis of the above research, the future work includes further extending typed bisimulation and completion of scheduler model.

REFERENCES

[1] H. P. Barendregt, "The Lambda Calculus, Its Syntax and Semantics," North Holland, Revised edition, 1985.
[2] C. A. R. Hoare, "Communicating Sequential Processes," Communication of the ACM, vol. 21, no. 8, 1978.
[3] U. Montanari and M. Pistore, "Concurrent Semantics for the Pi calculus," In: Proceedings of MFPS'95, Electronic Notes in Theoretical Computer Science, vol. 1, pp. 411-429, 1995.
[4] R. Milner, "Communication and Concurrency," Prentice-Hall, 1989, in press.
[5] R. M. Amadio, "An asynchronous Model of Locality, Failure, and Process Mobility," In: COORDINATION97, Springer LNCS vol. 1282, pp. 374–391, 1997.
[6] R. Devillers, H. Klaudel, M. Koutny and F. Pommereau, "Asynchronous Box Calculus," Fundamenta Informaticae, vol. 54, no. 4, pp. 295-344, 2003.
[7] E. Best, R. Devillers and J. G. Hall, "The Box Calculus: a New Causal Algebra with Multi-label Communication," Advances in Petri Nets 1992, vol. 609, pp. 21-69, 1992.
[8] S. Davide, "Locality and interleaving semantics in Pi calculus for mobile processes," Theoretical Computer Science, vol. 155, pp. 39-83, 1996.
[9] J. E. White, "Mobile Agents," MIT Press Cambridge, 1997, in press.
[10] J. C. M. Baeten and W. P. Weijland, "Process Algebra," Cambridge University Press, 1991, in press.
[11] R. Milner, J. Parrow and D. Walker, "A calculus of mobile processes part I/II," Journal of Information and Computation, vol. 100, no.1, pp.1-77, 1992.
[12] M. Boreale and D. Sangiorgi, "A fully abstract semantics of causality in the Pi calculus," Technical Report ECS-LFCS-94-297, 1994.
[13] J. Engelfriet, "A multiset semantics for the Pi-calculus with replication," Theoretical Computer Science vol. 153, pp. 65-94, 1996.

[14] B. Victor, "The Mobility Workbench User's Guider: Polyadic version 3.122," Department of Information Technology, Uppsala University, 1995.
[15] J. G. Jesse, "Ajax: A New Approach to Web Applications,"www.adaptivepath.com/publications/essays/archives/000385.php.
[16] C. Gross, "Ajax Patterns and Best Practices," Apress, 2006, in press.
[17] H. Klaudel and F. Pommereau, "Asynchronous links in the PBC and M-nets," In: Proceedings of ASIAN'99, Springer LNCS vol. 1742, pp. 190-200, 1999.
[18] K. Honda and M. Tokoro, "An Object Calculus for Asynchronous Communication," In: The 5th European Conference on Object-Oriented Programming, Springer LNCS vol. 512, pp. 133–147, 1991.
[19] M. Dam, "Proof Systems for Pi-Calculus Logics", Oxford University Press, 2001, in press.
[20] R. Milner, "Communication and Mobile Systems: The Pi-Calculus," Cambridge University Press, 1999, in press.

Hui Kang (1967-). Female, Han Dynasty, Changchun, Jilin Province, China, College of Computer Science and Technology in Jilin University. Vice Professor, Research Field: IT Services and Network management.

Zhi Wang (1986-). Male, Han Dynasty, Changchun, Jilin Province, China, College of Computer Science and Technology in Jilin University. Research Field: IT Services and Network management.

Shuangshuang Zhang (1989-). Female, Han Dynasty, Changchun, Jilin Province, China, College of Computer Science and Technology in Jilin University. Research Field: IT Services and Network management.

Fang Mei (1977-) Female, Changchun, Jilin Province, China, College of Computer Science and Technology in Jilin University. Research Field: IT Services and Network management.