Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

A PI-CALCULUS FORMALISM FOR DISCRETE EVENT SIMULATION

Jianrui Wang

Richard A. Wysk

Department of Industrial and Manufacturing Engineering

The Pennsylvania State University

University Park, PA 16802, USA

ABSTRACT

This paper presents PiDES, a formalism for discrete event simulation based on Pi-calculus. PiDES provides a rigorous semantics of behavior modeling and coordination for simulation federates. The capability of PiDES is demonstrated by translating a generalized semi-Markov process formalism into PiDES specification. The usage of PiDES is illustrated through a case study of a flexible manufacturing system consisting of two machines, two parts, and a robot. The major advantages of PiDES are discussed, which include: a) a complete set of semantics for both modeling and execution; b) supporting parallel and distributed simulation; c) adaptive modeling; d) rich coordination semantics for developing large simulation systems; and finally e) a formalism that can be used for agent-based simulation. An implementation of PiDES using Java programming language is also provided.

1 INTRODUCTION

Discrete Event Simulation (DES) is a powerful tool for modeling and controlling complex systems. For simulation purposes, target systems fall into two classes: quantitative analysis and qualitative analysis (Pooley 2007). For quantitative analysis, the emphasis is often the key performance indices (i.e., waiting time, queue length and resource utilization) of the systems. Therefore, stochastic models, such as *Generalized Semi-Markov Process* (GSMP) (Glynn 1989) become very useful. For qualitative analysis, the focus is on the behavior of the systems. Therefore, it is often more important to study how the components of the systems interact with each other. As a result, logical models, with minimum or no quantitative properties, such as *Finite State Automata* (FSA) (Ramadge and Wonham 1989), *Discrete Event System Specification* (*DEVS*) (Zeigler et al. 2000), *Petri Nets* (Zhou and Venkatesh 1999), and *process algebra* (D'Argenio et al. 1998; Harrison and Strulo 2000), have become quite popular for these types of models. These formalisms provide rigorous semantics and powerful mathematical tools for building and analyzing simulation models.

Although the above formalisms have proved useful for modeling individual systems, they become ineffective for some large scale complex adaptive systems. For instance, modern simulation federations often involve multiple participants, which are geographically distributed. Each modeling participant may implement simulation federates using different software technologies. In addition, existing federates may retire from the federation, while new federates may enter into the federation. This may require a control flow change of the simulation to reflect a frequently changing business model. Therefore, modern simulation federations have three unique characteristics: a) heterogeneous, b) distributed, and c) adaptive. As a result, formalisms for modern simulations should also provide companion semantics, such as: a) compositing heterogenic systems into larger ones; b) coordinating distributed systems; and c) evolving existing systems into new ones on the fly. Unfortunately, GMSP, FSA, and Petri Net formalisms provide little support for system composition. DEVS supports composition through coupled-DEVS, but is not fully adaptive. SPADE (Harrison and Strulo 2000) and ♠ (D'Argenio et al. 1998) may be improved to support many of the above features, but they are relatively new and need further study.

In this paper, we develop a formalism for DES based on *Pi-calculus* (Milner 1999), called PiDES (*Pi-calculus Discrete Event Simulation*). PiDES not only presents a set of formal models for modeling individual simulation federates, but also supports system composition and evolution. Furthermore, PiDES models are executable models, which can be automatically compiled into programming languages.

The remainder of this paper is organized as follows: Section 2 introduces the classical theory of Pi-calculus and

978-1-4244-2708-6/08/$25.00 ©2008 IEEE

its stochastic extensions; Section 3 presents the syntax of PiDES and its operational semantics based on an extended Pi-calculus; Section 4 demonstrates the modeling power of PiDES through translating GSMP models into PiDES models; Section 5 presents a case study of using PiDES to model a two-machine, two-part flexible manufacturing system; Section 6 briefly discusses the implementation of PiDES in Java programming language, The paper concludes in Section 7 with a discussion of PiDES and relative formalisms.

2 THEORETICAL FOUNDATION

This section introduces the theoretical foundation of PiDES – the classical theory of *Pi-calculus* developed by Milner *et al*. (Milner 1999) and its stochastic extensions from contemporary literatures.

A Pi-calculus process (*Pi-process*) is defined as one of four process as shown in Figure 1: 1) *summation*, 2) *composition*, 3) *restriction*, and 4) *replication*. A Pi-process has up to three capabilities: a) receives a name from a *channel* (i.e. $x(y)$ means receiving y though channel x.); b) sends a name by a channel (e.g. $\bar{x}(y)$ means sending y through channel x.); and c) performs an internal transition (called τ). The dynamics of Pi-process is formalized by a set of reduction rules (Figure 1):

- *TAU*. A summation Pi-process $\tau.P+M$ evolves to P by performing an internal transition τ. The inactive component (M) is discarded.
- *REACT*. A composition Pi-process conducts reduction through an interaction between two components that sharing the same channel name (one component send a name and the other receives a name through the same channel).
- *PAR*. If a Pi-process can evolve from P to P', then any composition Pi-process that has P as a directly composition component can perform a reduction.
- *RES*. Restricting a name does not affect internal reductions of a Pi-process.
- *STRUCT*. Two *structural congruence* (will be discussed shortly) Pi-processes (Milner 1999) have the same reduction behavior.

A powerful tool of studying the dynamics of Pi-calculus is *bisimulation* (Milner 1999). A bisimulation is a symmetric relationship between two processes (from now on, we use process and Pi-process interchangeably unless a distinguishing is necessary) that one can simulate the behavior of another step by step and vice versa. In general, two processes are considered equivalent (a.k.a. *structural congruence*) if they can bisimulate each other. In practice, structural congruence is obtained by applying any times of *α-conversion* (substituting a bounded name with a new name) and structural congruence rules defined in Figure 1. One important application of bisimulation and structural congruence is that any process can be converted into a standard form of a restricted composition process (Figure 1).

Pi-calculus Process:

$P \overset{def}{=} \sum_{i=1}^{m} M_i \,\|\, \prod_{i=1}^{n} P_i \,\|\, \nu x P \,\|\, !P$

$M_i \overset{def}{=} 0 \,\|\, \pi.P$

$\pi \overset{def}{=} x(y) \,\|\, \bar{x}(y) \,\|\, \tau$

$\sum_{i=1}^{m} M_i$	Summation, only one of M_i's ($1 \le i \le m$) will be performed and the others are void. The choice is non-deterministic.
$\prod_{i=1}^{n} P_i$	Composition, all P_i's ($1 \le i \le n$) run concurrently.
$\nu x P$	Restriction, the scope of name x is limited in P.
$!P$	Replication, there is infinite instances of P running concurrently.
0	An "inert process", which means no further actions.
$\pi.P$	Process P has to wait until π is performed.
$x(y)$	Receive y though channel x.
$\bar{x}(y)$	Send y through channel x.
τ	An internal transition.

Operational Semantics

TAU	$\tau.P + M$	$\rightarrow P$
REACT	$(x(y).P + M) \,\|\, (\bar{x}(z).Q + N)$	$\rightarrow P\{z/y\} \,\|\, Q$
PAR	$P \rightarrow P'$	$\rightarrow P \,\|\, Q \rightarrow P' \,\|\, Q$
RES	$P \rightarrow P'$	$\rightarrow \nu x P \rightarrow \nu x P'$
STRUCT	$P \rightarrow P', P \equiv Q, P' \equiv Q'$	$\rightarrow P' \rightarrow Q'$

α-conversion

$x(y).P = x(z).P\{z/y\}$	if z is not a free name of P.
$\nu x P = \nu w.P\{v/x\}$	if y is not a free name of P.

Structural Congruence

$P \,\|\, Q$	$\equiv$ $Q \,\|\, P$	
$(P \,\|\, Q) \,\|\, S$	$\equiv$ $P \,\|\, (Q \,\|\, S)$	
$\nu x P \,\|\, Q$	$\equiv$ $\nu x (P \,\|\, Q)$	if x is not a free name of Q.
$!P$	$\equiv$ $P \,\|\, !P$	
$P + Q$	$\equiv$ $Q + P$	

Standard Form

$P \equiv \nu(a_1, \cdots a_k)(M_1 \,|\, \cdots \,|\, M_m \,|\, !Q_1 \,|\, \cdots \,|\, !Q_n)$

All M_i's are non-empty summation process and Q_i's are also standard forms.

Figure 1: Pi-calculus syntax and operational semantics

The classical Pi-calculus is synchronized, that is, all transactions and reactions occur instantly without any delay. In addition, it only has a non-deterministic construct (choice) without probability. Recently, several stochastic

extensions have been proposed to handle probability and time duration (D'Argenio et al. 1998; Harrison and Strulo 2000; Priami 1995) by introducing new probability and timed constructs. In next section, a general theory of modeling systems using Pi-calculus and a formal definition of PiDES are proposed.

3 METHODOLOGY

This section presents the general theory of PiDES based on Pi-calculus.

Zeigler *et al*. proposed a *System Specification Hierarchy* (SSH) (Zeigler *et al*. 2000), which defines five levels of system specification: a) level 0 – observation frame, b) level 1 – I/O behavior, c) level 2 – I/O function, d) level 3 – state transition, and e) level 4 – coupled component. According to Zeigler *et al*., level 4 is the highest level definition in SSH. It models the components of a system and how they are coupled together. The components can be detailed at lower levels and form a hierarchical structure. In PiDES, a system is defined as a composition process, which is equivalent with level 4 definition of SSH. As discussed in previous section, any Pi-process is structural congruence to a composition process (a.k.a. standard form). Thus, a composition process is sufficient to represent any system that can be modeled by pi-calculus. The syntax of PiDES with its additional operational semantics is presented in Figure 2.

PiDES Process

$$P \overset{def}{=} \sum_{i=1}^{n}[\phi(p_i)]M_i \,\|\, \prod_{i=1}^{n} P_i \,\|\, \nu x P \,\|\, !P$$

$$M \overset{def}{=} 0 \,\|\, \pi.P$$

$$\pi \overset{def}{=} x(y) \,\|\, \overline{x}(y) \,\|\, [\lambda(f,r)]\tau \,\|\, [condition]\pi$$

Operational Semantics

$$PROB \quad \sum_{i=1}^{n}[\phi(p_i)]M_i \rightarrow M_i \qquad probability$$

$$WAIT \quad [\lambda(f,r)]\tau.P \rightarrow P$$

$$RACE \quad [\lambda(f_1,r_1)]\tau.P_1 + [\lambda(f_2,r_2)]\tau.P_2 \rightarrow P_1 \qquad if\ r_1 < r_2$$

$$\rightarrow P_1\ or\ P_2 \quad if\ r_1 = r_2$$

Figure 2: PiDES syntax and operational semantics

PiDES uses Pi-calculus channels to represent events. For example, x is an incoming event x and $\overline{x}$ is an outgoing event x in Figure 2. The data passing through these channels are attributes related to the event. There can be many attributes, but timestamp is the mostly interested one in this paper. In order to model system stochastic, PiDES introduces three extensions into the classical Pi-calculus:

1. $\sum_{i=1}^{n}[\phi(p_i)]M_i$ is a summation with probability choices, where $\phi(p_i)$ is the probability of choosing M_i and $\sum_{i=1}^{n}\phi(p_i) = 1$. $[\phi(p_i)]$ is optional, when it is omitted, each choice has the same probability to be rendered.
2. $[\lambda(f, r)]$ generates a random time delay r under the distribution function f. $[\lambda(f, r)]$ only applies to τ (the rationale is that delay is an internal transition). If no $[\lambda(f, r)]$ is specified, then the reaction occurs instantly.
3. [*condition*] decides whether the prefixed action is enabled. PiDES support common arithmetic operations (>, ≥, =, <, and ≤) and Boolean operations (AND, OR, and NOT). [*condition*] is also optional.

PiDES has all the operational semantics from classical Pi-calculus, plus three new operations shown in Figure 2. The first operation, *PROB*, provides a choice with probability. The second, *WAIT*, defines that action τ is enabled only after the associated clock runs down to 0. The last operation, *RACE*, indicates that two processes are competing for the opportunity to be executed. The process with a smaller clock value gets the chance to run, the other is void. If the two clocks have the same value, the result is undefined. In theory, such a non-deterministic behavior is fine. However, such a behavior often depends on runtime implementation and might not work as the designer expects. In practice, a conditional *RACE* (has condition or *PROB* for each component of *RACE*) is better to model non-deterministic behavior.

$$A \overset{def}{=} T_a + (T_r \mid B)$$

$$B \overset{def}{=} B_1 + B_2 + B_3$$

$$B_1 \overset{def}{=} \sum_{i=1}^{n}[\phi(p_i)][cond_i][\lambda(f_i,d_i)]\tau.(S(t+d_i, A_i) + B)$$

$$B_2 \overset{def}{=} \sum_{i=1}^{k} x_i(t_i).B_1$$

$$B_3 \overset{def}{=} \sum_{i=1}^{m} B_1.\overline{y}_i(t+d_i)$$

$$T_a \overset{def}{=} !ta(t_a).A\{t_a/t\}$$

$$T_r \overset{def}{=} !tr(y).\overline{y}(t)$$

$$S(t, A) \overset{def}{=} \nu ta(\overline{ta}(t) \mid A)$$

Figure 3: PiDES agent syntax

The essential building block for PiDES is *agent*, which is defined in Figure 3. An agent, *A,* has both a time-regulating model and a behavior model. The time-regulating model is fulfilled by two processes: T_a and T_r. T_a is a synchronizing process, which receives a time-advance event and advances A's logical time. It is critical that when time-advance is performed, any other capability of A is temporarily deactivated in order to avoid time-

inconsistency (this is similar to concurrent programming, a monitor is used to guard shared data in order to avoid data inconsistency). T_r offers a query interface for publishing A's current logical time through a new channel provided by the requester. It is clear that performing T_r does not disable any of A's capability. The behavior model of A is provided by process B. It defines A's three possible behaviors: 1) performs an internal transition, then evolves to a new agent A_i (time advanced by $S(t, A)$) or performs a behavior model with probability p_i and condition $[cond_i]$ (i.e. a time constraint for ignoring past-time events); 2) receives a new event x_i, then performs behavior B_1; and 3) performs behavior model B_1 then sends out an event y_i. It should be noted that, A_i can be any agent including A itself, which may be used to model memory-less agent. The time advance action is formalized as process $S(t, A)$. It first creates a private channel *ta*, then send a new time t to process A, which also has a channel *ta* with receiving capability. Once A receives a new time t, it performs a α-conversion to update its logical time to t.

The second building block of PiDES is *coordination context* as shown in Figure 4. A *process context* is a process with place-holdings for other processes (Milner 1999). The classical context Pi-calculus is defined as C_π in Figure 4. C_π provides the capability of compositing many processes in to a larger process. However, it does not provide much information about how to coordinate these processes. This process context concept is extended in PiDES into coordination context, which provides coordination mechanisms for multiple agents and simulation federates. In order to facilitate interactions between agents and federates, PiDES offers two coordination contexts: *Orchestration Coordination Context* (OCC) and *Choreography Coordination Context* (CCC).

$$
\begin{aligned}
C &\overset{def}{=} C_o \,\|\, C_c \\
C_o &\overset{def}{=} C_\pi \,\|\, C_{seq} \,\|\, C_{prob} \,\|\, C_{dec} \,\|\, C_{loop} \\
C_{seq} &\overset{def}{=} \nu(a_1, a_2, ..., a_n)(C.\bar{a}_1 \mid \bar{a}_1.C \mid ... \mid \bar{a}_{n-1}.C) \\
C_{prob} &\overset{def}{=} \sum_{i=1}^{n} \phi(p_i)C \\
C_{dec} &\overset{def}{=} \sum_{i=1}^{n} [condition_i]C \\
C_{loop} &\overset{def}{=} [condition]\tau.C + [NOT\ \ condition]\tau.0 \\
C_c &\overset{def}{=} \coprod_{i=1}^{n} C_i = C_1 \mid C_2 \mid \cdots \mid C_n \\
C_\pi &\overset{def}{=} [\ ] \mid \pi.C_\pi + M \,\|\, \nu x C_\pi \,\|\, (C_\pi \mid P) \,\|\, !C_\pi
\end{aligned}
$$

Figure 4: PiDES coordination context

An OCC C_0 provides a control flow based coordination mechanism. C_0 is one of five contexts. C_π is the classical process context. C_{seq} is a *sequential context*, where all the sub-contexts are executed sequentially. C_{prob} is a *probability context* that performs one of its sub-contexts with probability. C_{dec} is a *decision context*, where a sub-context will be performed only if the associated the condition is hold. C_{loop} is a *loop context*, where the sub-context is performed repeatedly until the associated condition is no longer hold. Similar to programming languages (i.e., C, C++, and Java), C_{seq}, C_{dec}, and C_{loop} are sufficient to build any sequential control flow. A CCC C_c is a coordination context that each sub-context can freely interacts with others. Therefore, it is defined as a parallel composition.

After formalize agent and coordination context, it is easy to define a simulation system in PiDES. A simulation system is simply a CCC. That is, a simulation system is modeled as a parallel system that includes many concurrent running sub-systems.

In next section, we show that PiDES is indeed sufficient to model DES by translating GSMP specification into PiDES specification.

4 VALIDATION

In this section, we prove PiDES is capable of modeling DES. The proof is done by using PiDES to bisimulate GSMP. As discussed in Section 2, bisimulation is an equivalent relation between *state transition systems*. Two systems are bisimilar if they match each other's every move. Thus, the behavior of these systems cannot be distinguished from an outside observer (Milner 1999). If PiDES can bisimulate GSMP, then PiDES has the equivalent modeling power of GSMP. Since GSMP is a formalism of DES, then PiDES is also a formalism of DES.

A GSMP is a 6-tuple (S, s_0, E, P, C, F) (Glynn 1989), where:

- S is a set all the states.
- s_0 is the initial state, $s_0 \in S$.
- E : is the set of all events.
- $P : s_i \times e \rightarrow s_{i+1}$, is a set of probability of jumping from state s_i to s_{i+1}. triggered by event e, $e \in E$.
- C : is a set clocks, each clock is corresponding to a state s and an event e. It continues to count down to zero.
- F: is the probability function of scheduling a new event e' in state s', given the previous state is s and the triggered event is e.

In PiDES, there are no states, as each state in GSMP is transferred into an agent. GSMP events are modeled as channels pairs (one for sending and the other for receiving). The probability jumping function P is modeled as behavior model B_2 in Figure 3. Clocks in C are presented by $[\lambda(f, r)]$. Finally, probability functions of scheduling new events in F is modeled as B_3 in Figure 3. Thus, a GSMP model can be bisimulated by a PiDES model

through the following steps:

1. Each state of GSMP becomes a PiDES agent.
2. The agent corresponding to s_0 is the starting agent for the system.
3. Each event becomes a pair of channel names. One for receiving and the other for sending.
4. All the jumping relation in P related to s_i are combined into the behavior model B_2 of agent A_i.
5. Clocks in C are modeled as B_1 in each agent.
6. All the probability functions of scheduling new events related to s_i are combined into the behavior model B_3 of agent A_i.

Readers may find that the above process does not involve any coordination context. This is because GSMP has no explicit semantics for coordination. The execution of GSMP models is determined by transition rules, which are implicitly captured by PiDES's CCC. It should also be noted, although the above algorithm translates each state into an agent, it is generally unnecessary and inefficient for PiDES. In fact, PiDES prefers *Agent-Based Modeling* (ABM)(Gilbert 2008; Macal and North 2005) perspective due to the nature that PiDES has rich behavior models. In the next section, we show how to model a two-machine, two-part FMS using PiDES.

5 CASE STUDY

In this section, a simple PiDES model is presented. It models a two-machine, two –part Flexible Manufacturing System (FMS) (Chang et al. 1998) as shown in Figure 5. This FMS uses one robot and two machines to process two parts. Part 1 is send to machine A first, processed for 4 unit time, then send to machine B for another 5 unit time processing. Part 2 is send to machine B first, then to machine A, the processing time 5 unit and 4 unit respectively. All the material handlings are done by the robot.

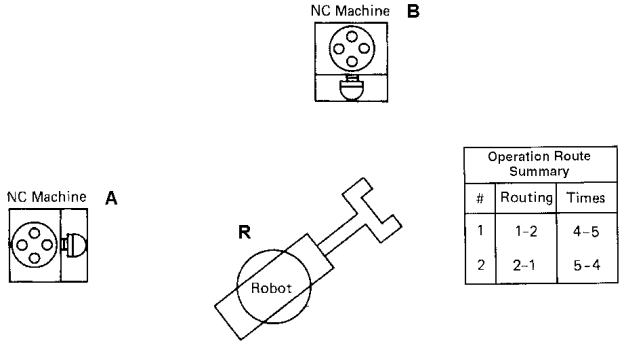

Figure 5: A two-machine, two-part FMS (Chang et al. 1998)

As discussed in Section 4, PiDES uses ABM perspective. Three agents are identified: machine A, machine B, and robot R. The system (F) is modeled in a CCC with three agents A, B and R as shown in Figure 6.

$$F \overset{def}{=} A \mid B \mid R$$

Figure 6: PiDES model for a two-machine, two-part FMS

Figure 7 shows the definition of machine A. It has 6 behavior categorized in three categories: (1) processes part (B_1^1 and B_1^2), which first performs a constant time delay, then transits to model B_3; (2) receives part (B_2^1 and B_2^2) through event r (the first subscript denotes part number and the second denotes the part state), then perform B_1; and (3) finishes part (B_3^1 and B_3^2) and starts a new cycle with performing B_1^0. The models of agent B and robot R can be developed in a way similar to A, which are shown in Figure 8 and Figure 9 respectively.

$A \overset{def}{=} T_a + (T_r \mid (B_2^1 + B_2^2))$	Agent definition
$B_1^1 \overset{def}{=} [\lambda(4,d)]\tau.B_3^1\{(t+d)/t\}$	Process part 1
$B_1^2 \overset{def}{=} [\lambda(5,d)]\tau.B_3^2\{(t+d)/t\}$	Process part 2
$B_2^1 \overset{def}{=} r_{1,0}(t).B_1^1$	Receive part 1
$B_2^2 \overset{def}{=} r_{2,1}(t).B_1^2$	Receive part 2
$B_3^1 \overset{def}{=} \bar{a}_{1,1}(t).B_1^0$	Finish part 1
$B_3^2 \overset{def}{=} \bar{a}_{2,1}(t).B_1^0$	Finish part 2
$B_1^0 \overset{def}{=} \tau.A$	Continue

Figure 7: PiDES models for machine A

$B \overset{def}{=} T_a + (T_r \mid (B_2^1 + B_2^2))$	Agent definition
$B_1^1 \overset{def}{=} [\lambda(5,d)]\tau.B_3^1\{(t+d)/t\}$	Process part 1
$B_1^2 \overset{def}{=} [\lambda(4,d)]\tau.B_3^2\{(t+d)/t\}$	Process part 2
$B_2^1 \overset{def}{=} r_{1,1}(t).B_1^1$	Receive part 1
$B_2^2 \overset{def}{=} r_{2,0}(t).B_1^2$	Receive part 2
$B_3^1 \overset{def}{=} \bar{b}_{1,2}(t).B_1^0$	Finish part 1
$B_3^2 \overset{def}{=} \bar{b}_{2,1}(t).B_1^0$	Finish part 2
$B_1^0 \overset{def}{=} \tau.B$	Continue

Figure 8: PiDES models for machine B

The execution of F is performed through reductions using PiDES operations presented in Figure 1 and Figure 2. It is obvious that F will run to deadlock quickly. The reason is that both machine A and B are competing on a single resource robot R and no coordination context is provided in the target system. The full deadlock analysis and

solution can be found in (Chang et al. 1998). From a system perspective, two courses of actions are available to deal with deadlocks: (a) change the behavior of each agent and (b) provide coordination context for the agents. For instance, the deadlock caused by sending event $\bar{r}_{1,0}$ and $\bar{r}_{2,0}$ (or $\bar{r}_{2,0}$ and $\bar{r}_{1,0}$) in a row can be avoid by changing the behavior B_3^5 and B_3^6 of *R* as shown in Figure 10.

$R \stackrel{def}{=} T_a + (T_r \mid (\sum_{i=1}^{4} B_2^i + \sum_{j=1}^{6} B_3^j)$	Agent definition
$B_1^1 \stackrel{def}{=} \tau.B_3^1$	Handling part 1
$B_1^2 \stackrel{def}{=} \tau.B_3^2$	Handling part 2
$B_1^3 \stackrel{def}{=} \tau.B_3^3$	Handling part 1
$B_1^4 \stackrel{def}{=} \tau.B_3^4$	Handling part 2
$B_2^1 \stackrel{def}{=} a_{1,1}(t).B_1^1$	Receive part 1 from A
$B_2^2 \stackrel{def}{=} a_{2,2}(t).B_1^2$	Receive part 2 from A
$B_2^3 \stackrel{def}{=} b_{1,2}(t).B_1^3$	Receive part 1 from B
$B_2^4 \stackrel{def}{=} b_{2,1}(t).B_1^4$	Receive part 2 from B
$B_3^1 \stackrel{def}{=} \bar{r}_{1,1}(t).B_1^0$	Send part 1 to B
$B_3^2 \stackrel{def}{=} \bar{r}_{2,2}(t).B_1^0$	Send part 2 to storage
$B_3^3 \stackrel{def}{=} \bar{r}_{1,2}(t).B_1^0$	Send part 2 to storage
$B_3^4 \stackrel{def}{=} \bar{r}_{2,1}(t).B_1^0$	Send part 1 to A
$B_3^5 \stackrel{def}{=} \bar{r}_{1,0}(t).B_1^0$	Send part 1 to A
$B_3^6 \stackrel{def}{=} \bar{r}_{2,0}(t).B_1^0$	Send part 2 to B
$B_1^0 \stackrel{def}{=} \tau.R$	Continue

Figure 9: PiDES models for robot *R*

However, the above approach might be constraining for large systems. When the number of agents in the systems becomes large, it is unlikely to avoid deadlocks by only changing behavior of individual agent. Thus, action (b) becomes more useful. The PiDES coordination contexts, especially OOC, are useful to prevent deadlock. However, deadlock problem is very complicated and it is not the main purpose of this paper. Design deadlock-free systems based on Pi-calculus can be found in (Kobayashi 2006).

$B_3^5 \stackrel{def}{=} \bar{r}_{1,0}(t).B_2^1$	Send part 1 to A
$B_3^6 \stackrel{def}{=} \bar{r}_{2,0}(t).B_2^4$	Send part 2 to B

Figure 10: Behavior change for avoiding deadlock

Another common situation for systems modeling is how to adapt the models when changes are requested. This problem can be solved by PiDES simply. For example, suppose we have a new *B* machine (*B'*), then the system F can be adapted to a new one (*G*) by simply adding a new parallel agent *B'* as shown Figure 11. There is no other changes required for the model.

$$G \stackrel{def}{=} F \mid B$$
$$= A \mid B \mid B' \mid R$$

Figure 11: System evolution with a new machine *B'*

6 IMPLIMENTATION

The implementation of PiDES consists of two parts: a) a compiler that reads, validates, and translates PiDES models into code that understood by a computer (e.g. java code) and b) a *Runtime Infrastructure* (RTI) that executes the computer code. The detail discussion of the RTI is beyond the scope of this paper. Here, we briefly describe the implementation of compiling PiDES models into Java code.

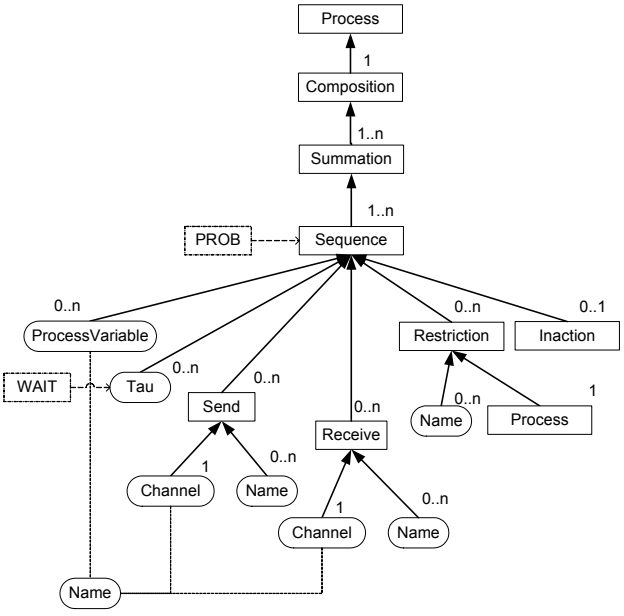

Figure 12: PiDES grammar tree

The syntax of PiDES presented in Figure 2 is ambiguous (i.e. a simple process such as x(y) can be interpreted as either a summation or a composition) and cannot be used to construct a compiler directly. Thus, the first step is to make the syntax unambiguous. Figure 12 provides an unambiguous grammar equivalent to the definition shown in Figure 2, which is used in this paper. As can be seen from Figure 12, a PiDES process is strictly defined in a hierarchy of PiDES terms. Therefore, for any given PiDES process, there is a unique parsing tree from the root (Proc-

ess) to the leaves (Names). Figure 12 also shows that replication is removed from the grammar. This is mainly for convenience because infinity is hard to implement. Furthermore, a replication process is structural congruence to a recursion (Milner 1999). Therefore, PiDES implements replication through recursion, which is obtained though *ProcessVariable* (an alias of a process).

The second step is to translate PiDES models into a compiler recognizable literal stream. PiDES uses symbols that are not support by most programming language (i.e. Java). It has to be encoded before being fed to the compiler. In this paper, an ASCII encoding similar to (Li 2005) is used. Table 1 shows how some PiDES terms are encoded.

Table 1: ASCII encoding for PiDES terms

PiDES term	ASCII encoding
$x(y)$	in(x, y)
$\bar{x}(y)$	out(x, y)
τ	tau()
vx	new(x)
$\phi(p)$	prob(p)
$\lambda(f, r)$	delay(f, r)

The ASCII encoded PiDES models are compiled into Java code by the PiDES compiler. The compiler is developed in *Java* using *JavaCC* (JavaCC 2007). The detail information about the BNF grammar of PiDES can be found in Appendix A.

7 DISCUSSION

This paper presents PiDES, a new formalism for DES. The subject of formalizing DES has been studied extensively (Smith 2003). From early formal models based on FSA (Ramadge and Wonham 1989) and stochastic process (Glynn 1989), to DEVS (Zeigler et al. 2000), to Petri Net based (Zhou and Venkatesh 1999), and to recent process algebra based formalisms (D'Argenio et al. 1998; Harrison and Strulo 2000). The focus of modeling simulation systems has been gradually shifting from quantitative analysis to behavioral analysis due to the fast increasing of system complexity and requirement of adaptive structures. Some latest development in simulation area might be Agent-Based Simulation (Gilbert 2008). However, ABS is often presented as software library such as SWARM and Repast (Gilbert 2008) without rigorous formalisms.

Formal approach for modeling system behavior has also been well studied, such as general theory of action and time (Allen 1984) and behavior equivalence in simulation modeling (Pooley 2007). Recently, many system and behavior modeling research have been using process algebra, especially Pi-calculus (Milner 1999). Researches about building deadlock-free systems using Pi-calculus can be found in literatures (Kobayashi 2006). PiDES is an extension of conventional DES formalisms with integration of behavior modeling power provided by Pi-calculus. A comparison of PiDES and other DES formalisms is shown in

Table 2.

Table 2: Comparison of DES Formalisms

Formalism	**Semantics**	**Concurrency**	***compositionality***
FSA	Strong	Weak	*Weak*
Petri Net	Strong	Strong	*Weak*
GSMP	Strong	Weak	*Weak*
DEVS	Strong	Strong	*Good*
PiDES	*Strong*	*Strong*	*Strong*

As can be seen from

Table 2, PiDES has better support of concurrency and compositionality than other formalisms. These two advantages make PiDES more suitable for modeling large parallel and distributed discrete event simulation systems. Therefore, PiDES can also serve as a theoretical foundation for to *High Level Architecture* (HLA) (IEEE et al. 2000). In a word, the major advantages of PiDES are: a) a complete set of semantics for both modeling and execution; b) supporting parallel and distributed simulations; c) models are adaptive; d) rich coordination semantics for developing large simulation systems; and e) a formalism that can be used for Agent-Based Simulation. As a complement, the authors are working on using PiDES to provide semantics for HLA *Runtime Infrastructure* (RTI) (IEEE et al. 2000).

ACKNOWLEDGEMENT

This research was partially inspired by Dr. Albert Jones, director of the Enterprise Systems Group at NIST, in communication with the second author about modeling complex systems. Some of the early ideas have been presented on IERC 2007 under title "*A Framework for Simulation-Based Shop Floor Control with Evolving Structure*" and a guest lecture at NIST under title "*A Unified Framework Using Pi-calculus for Modeling and Coordinating Large Complex Systems*", 2007.

A PIDES BNF GRAMMAR (WITHOUT TERMINALS)

```
        Federate ::= <FEDERATE> PiName "{" ( Agent )+ "}" <EOF>
           Agent ::= <AGENT> PiName "=" Composition ";"
     Composition ::= Summation ( <COMP> Summation )*
       Summation ::= ( Sequence ( <SUM> Sequence )* )
                   | ( Probability Sequence ( <SUM> Probability Sequence )* )
        Sequence ::= AtomicProcess ( <DOT> AtomicProcess )*
  AtomicProcess ::= PiName
                   | Inaction
                   | Action
                   | "(" Composition ")"
                   | RestrictionProcess
RestrictionProcess ::= <NEW> "(" PiNameList ")" "(" Composition ")"
          Action ::= Tau
                   | Send
                   | Recieve
            Send ::= ( Condition )? <OUT> "(" PiNameList ")"
         Recieve ::= <IN> "(" PiNameList ")"
             Tau ::= ( Delay )? <TAU>
        Inaction ::= <INACTION>
      PiNameList ::= ( PiName ( "," PiName )* )?
          PiName ::= <PINAME>
       Condition ::= "[" <PINAME> ComparisonOp <PINAME> "]"
    ComparisonOp ::= ">"
                   | ">="
                   | "<"
                   | "<="
                   | "="
                   | "!="
     Probability ::= <PROB> "(" <PINAME> ")"
           Delay ::= <DELAY> "(" Distribution ")"
    Distribution ::= DistNorm
                   | DistConst
                   | DistPoisson
        DistNorm ::= <NORM> "(" <NUMBER> "," <NUMBER> "," <PINAME> ")"
       DistConst ::= <CON> "(" <NUMBER> "," <PINAME> ")"
     DistPoisson ::= <POISSON> "(" <NUMBER> "," <NUMBER> "," <PINAME> ")"
```

REFERENCES

Allen, J. F. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2): 123-154.

Chang, T.-C., R. A. Wysk, and H. P. Wang. 1998. *Computer-aided manufacturing*. Upper Saddle River, N.J.: Prentice Hall.

D'Argenio, P. R., J-P. Katoen, and E. Brinksma. 1998. General purpose discrete-event simulation using SPADES. *Proceeding of 6th Process Algebra and Performance Modeling Workshop*, ed. C. Priami, 85-102, Nice, France: Universit di Verona.

Gilbert, G. N. 2008. *Agent-based models*. Los Angeles: Sage Publications.

Glynn, P. W. 1989. A GSMP formalism for discrete event systems. *Proceedings of the IEEE* 77(1): 14-23.

Harrison, P. G. and Strulo, B. 2000, SPADES - a process algebra for discrete event simulation. *Journal of Logic and Computation* 10(1): 3-42.

IEEE, et al. 2000. *IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA) : framework and rules*. New York, NY: Institute of Electrical and Electronics Engineers.

JavaCC. 2007. JavaCC 4.0. Retrieved 2/25/2008, from https://javacc.dev.java.net/.

Kobayashi, N. 2006, A New Type System for Deadlock-Free Processes. *CONCUR 2006 – Concurrency Theory*, 233-247: Springer-Verlag New York, Inc.

Li, L. 2005. An Implementation of the Pi-Calculus on the .NET. *Journal of Object Technology* 4(5): 20.

Macal, C. M. and North, M. J. 2005. Tutorial on agent-based modeling and simulation. *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, 2-15. Orlando, FL: IEEE Inc, Piscataway, NJ.

Milner, R. 1999. *Communicating and mobile systems: the pi-calculus*. New York: Cambridge University Press.

Pooley, R. 2007. Behavioural equivalence in simulation modelling. *Simulation Modelling Practice and Theory* 15(1): 1-20.

Priami, C. 1995. Stochastic Pi-Calculus. *The Computer Journal* 38(7): 578-589.

Ramadge, P. J. G. and W. M. Wonham. 1989, The control of discrete event systems. *Proceedings of the IEEE* 77(1): 81-98.

Smith, J. S. 2003. Survey on the use of simulation for manufacturing system design and operation. *Journal of manufacturing systems* 22(2): 157-171.

Zeigler, B. P., T. G. Kim, and H. Praehofer. 2000. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. San Diego: Academic Press.

Zhou, M. and K. Venkatesh. 1999, *Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach*. River Edge, NJ: World Scientific.

AUTHOR BIOGRAPHIES

JIANRUI WANG is a Ph.D. candidate at Department of Industrial and Manufacturing Engineering, The Pennsylvania State University at University Park. He received a M.S. degree in Business Administration with concentration on Supply Chain and Information Systems from Smeal College of Business at Penn State University. He also holds a M.S. degree in Mechanical Engineering, a B.S. degree in Manufacturing Engineering, and a dual B.S. degree in Computer Engineering, all from Shanghai Jiao Tong University, China. His current research interests are discrete event simulation, complex adaptive systems, business process management, supply chain management, and service-oriented architecture. He can be contacted by email at <jerrywang@psu.edu>.

RICHARD A. WYSK is the Leonhard Chair in Engineering and a Professor of Industrial Engineering at Pennsyl-

vania State University, University Park. Dr. Wysk has co-authored six books including *Computer-Aided Manufacturing*, with T.C. Chang and H.P. Wang -- the 1991 IIE Book of the Year and the 1991 SME Eugene Merchant Book of the Year. He has also published more than a hundred and fifty technical papers in the open-literature in journals including the *Transactions of ASME*, the *Transactions of IEEE* and the *IIE Transactions*. He is an Associate Editor and/or a member of the Editorial Board for five technical journals. Dr. Wysk is an IIE Fellow, an SME Fellow, a member of Sigma Xi, and a member of Alpha Pi Mu and Tau Beta Pi. He is the recipient of the IIE Region III Award for Excellence, the SME Outstanding Young Manufacturing Engineer Award and the IIE David F. Baker Distinguished Research Award. He has held engineering positions with General Electric and Caterpillar Tractor Company. He received his Ph.D. from Purdue University in 1977. He has also served on the faculties of Virginia Polytechnic Institute and State University and Texas A&M University where he held the Royce Wisenbaker Chair in Innovation. His can be contacted by email at <rwysk@psu.edu>.