Relating π-calculus to Object-Z

Kenji Taguchi
Dept. of Computing, Univ. of Bradford, UK
K.Taguchi@bradford.ac.uk

Jin Song Dong
Computer Science Dept., Nat'l Univ. of Singapore, Singapore
dongjs@comp.nus.edu.sg

Gabriel Ciobanu
Romanian Academy, Inst. Computer Science, Iaşi, Romania
gabriel@iit.tuiasi.ro

Abstract

Software systems have become increasingly distributed, dynamic and mobile. The complex state and dynamic interfaces of software components and their concurrent interactions provide challenging research issues in system specification and design. An effective combination of structured state-based formalism and dynamic action-based calculus may be a good solution for modeling complex distributed mobile systems. In this paper, we investigate the semantic links between Object-Z and π-calculus and consequently introduce a powerful specification technique PiOZ that brings the strengths of the two together. The operational semantics of PiOZ integrates state transition semantics of Object-Z and π-calculus reduction rules. The typing rules of PiOZ are developed and reasoning of a system property is presented.

1. Introduction

Software systems have become increasingly distributed and mobile. The complex state and dynamic interfaces of software components and their concurrent interactions provide challenging research issues in system design and implementation. Although some specification techniques can support dynamic communications and mobility, such as π-calculus [18, 17], they generally cannot scale up for modeling the data and state of complex distributed systems. On the other hand, state-based formalisms such as Z [31] and Object-Z [7, 23], are powerful specification techniques for capturing complex data and states; however, they are weak for capturing the dynamic communication configurations for distributed systems. The usual technique for modeling complex systems is the multi-viewpoints approach [10, 19]. However, the problem of consistency issues between different viewpoint models (represented in different formalisms) can only be clearly resolved by working out the semantic link/integration between those meta notations.

An effective combination of structured state-based formalism and dynamic action-based calculus may be a good solution for modeling complex distributed mobile systems. In this paper, we investigate the semantic links between Object-Z and π-calculus and consequently introduce a powerful specification technique PiOZ that brings the strengths of the two together. In our approach, the fundamental semantic link is to treat the Object-Z operations and π-calculus processes as the same semantic and syntactic entity. The consequence of this semantic link is that complex operations can be defined by composing simple operation schemas by using π-calculus process operators (parallel, choice and sequential). Furthermore, we enforce that all communications between objects are through type safe π-calculus channels. The communication interface of PiOZ objects are defined by channel-typed state variables (treated the same as other state variables) so that the dynamic communication interface change is captured by the channel-typed variables' state change. For instance, the π-calculus input guard (binding operator) is given a state-update semantics. As a result, the formal operational semantics of PiOZ integrates state transition semantics of Object-Z and π-calculus reduction rules, providing a foundation for derivation and reasoning system properties in PiOZ. From the state point of view, PiOZ extends Object-Z with π-style dynamic communication capabilities. From the event point of view, PiOZ class model contains all possible π-style instance level reductions; that is one π-style reduction represents one possible state-change trace (animation) of the PiOZ object's behaviour. The typing rules of PiOZ can also be constructed based on π-calculus and Object-Z typing rules.

The paper is organised as follows. First, π-calculus and Object-Z are briefly introduced. The advantages and disadvantages of the two notations in modeling data/state and communication aspects are illustrated by using a common example, message passing between stations and a car (an extended and modified version of Milner's mobile phone system [17].) The semantic links between π-calculus and Object-Z are discussed, the integrated notation PiOZ is introduced. In Chapter 4, typing rules of PiOZ are developed and Section 5 presents the integrated PiOZ operational semantics with a derivation (π-style reduction). Chapter 6 presents a verification of a system property for the PiOZ model. Finally, a discussion of related work and conclusion is presented.

2. π-calculus and Object-Z

In this section we provide overviews for the π-calculus and Object-Z.

2.1. π-calculus

The π-calculus is a widely accepted model of interacting systems with dynamically evolving communication topology. π-calculus allows channels to be passed as data along other channels, and this introduces a channel mobility. An important feature of the π-calculus is its mobility expressed by the changing configuration and connectivity among processes. This mobility increases the expressive power enabling the description of many high-level concurrent features. π-calculus has a simple semantics and a tractable algebraic theory.
The computational world of the π-calculus contains just processes (also called agents) and channels (also called names or ports). π-calculus models networks in which messages are sent from one site to another site and may contains links to active processes or to other sites. π-calculus is a general model of computation which takes interaction as primitive. However the π-calculus is not suitable to describe state changes.
We present in this section the monadic version of the π-calculus: this means that a message consists of exactly one name. Let $\mathcal{X}$ be a infinite countable set of *names*. The elements of $\mathcal{X}$ are denoted by $x, y, z \ldots$ The terms of this formalism are called processes and processes are denoted by $P, Q, R \ldots$.

Definition 1 *The* **processes** *are defined over the set $\mathcal{X}$ of names by the following grammar*

$$P ::= 0 \mid \overline{x}\langle z\rangle.P \mid x(y).P \mid P \mid Q \mid P + Q \mid !P \mid \nu x\, P$$

The process expressions are defined by guarded processes $\overline{x}\langle z\rangle.P$ and $x(y).P$, parallel composition $P \mid Q$, nondeterministic choice $P + Q$, replication $!P$ and a restriction $\nu x\ P$ creating a local fresh channel x for the process P. 0 is the empty process. π-calculus replication $!P$ can also be expressed by recursive equations of parametric processes. The guards are input guards and output guards. They represent sending and receiving a message (name) along a channel. The output guarded process $\overline{x}\langle z\rangle.P$ sends z along x and then, after the output has completed, continues as P. An input guarded process $x(y).Q$ waits until a name is received along x, substitutes it for the bound variable y and continues as Q. The parallel composition $\overline{x}\langle z\rangle.P \mid x(y).Q$ may thus synchronize on x. Thus processes can interact by using names they share. A name received in one interaction can be used in another; by receiving a name, a process can interact with processes which are unknown to it, but now they share the same channel name. The π-calculus mobility is coming from its scoping of names and extrusion of names from their scopes.
There is an important distinction between input and output guards. Output guard is a simple sending of a name z along a channel x, but the input guard has a more complex action: the name received along the channel x will replace y in the process following the input guard. Input guard is a *binding* operator involving substitutions. In $x(y).P$, the name y binds free occurrences of y in P. In a second binding operator $\nu x\ P$, the name x binds free occurrences of x in P.
Over the set of processes a structural congruence relation is defined; this relation provides a static semantics of some formal constructions. We denote by $fn(P)$ the set of the names with free occurrences in P, and by $=_\alpha$ the standard α-conversion.

Definition 2 *The relation $\equiv$ over the set of processes is called structural congruence, and it is defined as the smallest congruence which satisfies*

- $P \equiv Q$ *if* $P =_\alpha Q$
- $P + 0 \equiv P$, $P + Q \equiv Q + P$, $(P + Q) + R \equiv P + (Q + R)$,
- $P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$,
- $!P \equiv P \mid !P$
- $\nu x 0 \equiv 0$, $\nu x \nu y P \equiv \nu y \nu x P$,
  $\nu x(P \mid Q) \equiv P \mid \nu x Q$ *if* $x \notin fn(P)$.

The structural congruence deals with the aspects related to the structure of the processes. The evolution of a process is described in π-calculus by a reduction relation over processes called reaction. This reaction relation contains those transitions which can be inferred from a set of rules.

Definition 3 *The reduction relation over processes is defined as the smallest relation $\rightarrow$ satisfying the following*

rules

$$
\begin{array}{ll}
(com) & (\overline{x}\langle z\rangle.P + R_1) \mid (\,x(y).Q + R_2) \rightarrow P \mid Q\{z/y\} \\
(par) & P \rightarrow Q \text{ implies } P \mid R \rightarrow Q \mid R \\
(res) & P \rightarrow Q \text{ implies } (\nu x)P \rightarrow (\nu x)Q \\
(str) & P \equiv P',\ P' \rightarrow Q' \text{ and } Q' \equiv Q \text{ implies } P \rightarrow Q
\end{array}
$$

The most studied forms of behavioural equivalence in process algebras are based on the notion of bisimulation. There are several definitions in the literature for bisimilarity; one of them is called open bisimilarity. Its definition is given by using the labelled transition system defined by the reduction rules. Systems can be checked automatically by studying the bisimilarity between two processes, namely the model and its specification. More helpful in the verification process is the *weak open bisimilarity*. It allows the basic verification technique for proving properties about mobile concurrent systems modeled in the π-calculus. In general, properties of finite state transition systems can be described in a very powerful logic called μ-calculus. Modeling and verifying with this logic and some of its proper subsets have been thoroughly investigated in [5]. Model checking π-calculus Mobility Workbench[29] supports open bisimulation checking.

Mobility: An example

The π-calculus is able to describe mobile systems, providing a conceptual framework and mathematical tools. The word mobility is used with many meanings. The π-calculus deals with the mobility given by links that move in a space of linked processes. For example, hypertext links can be created, can be passed around and can disappear. Or references can be passed as arguments of method invocations in object-oriented systems. Our example describes a simple interaction between a handphone carried in a car and some base stations. The system is described in Figure 1. The connections between our car (handphone) and the base stations can change as the car is moving around.
We consider three processes B_1, B_2 and C corresponding to two bases and the car respectively. We start with their parallel composition $B_1 \mid C \mid B_2$ described by the left picture of Figure 1. The base B_1 and car C are connected by a channel *talk* and B_1 and B_2 by a channel *switch*. This means that *talk* is free in both B_1 and C, and *switch* is a free name in both B_1 and B_2. By the process expression $\nu\ talk\ (B_1 \mid C) \mid B_2$, the name *talk* is restricted to B_1 and C, and we interpret that B_1 and C have an exclusive communication along the channel *talk*. If $B_1 = \overline{switch}\langle talk\rangle.B_1'$, then base B_1 wishes to send the name of channel *talk* to base B_2 along the channel *switch*. Moreover, if *talk* is not free in B_1' ($talk \notin fn(B_1')$), then B_1' will lose its link to C. Base B_2 is waiting for a channel name sent by B_1, namely $B_2 = switch(y).B_2'$. Applying the corresponding reaction rules, we have the transition

$$\nu\ talk\ (B_1 \mid C) \ \mid\ B_2 \ \longrightarrow\ B_1' \ \mid\ \nu\ talk\ (C \mid B_2'')$$

where $B_2'' = B_2'\{talk/y\}$.
The initial process $\nu\ talk(B_1 \mid C) \mid B_2$ changes its communication topology and it becomes as it is described in Figure 1. Now B_2'' and C have an exclusive communication along the channel *talk*. This is essentially the mobility mechanism offered by the π-calculus. More details are in [17].

Difficulty with state/system capturing

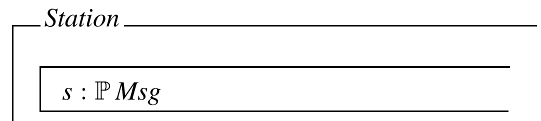
Note that π-calculus is an excellent tool for capturing a particular behaviour at instance level (e.g. the example above only captures a partial behaviour trace of a communication switch between two base stations and a car. However, one may consider an arbitrary group of bases (instead of just two bases), and *talk* channel passed back and forth between these bases. More importantly, one may consider the state changes of base stations and the car (e.g. when a base station passes a message to the car, that base should have one less message in its collection and the car should have one more message). These requirements are difficult or impossible to be captured by the π-calculus.

2.2. Object-Z Overview

Object-Z is an extension of the Z formal specification language to accommodate object orientation. The main reason for this extension is to improve the clarity of large specifications through enhanced structuring.

Class

The essential extension to Z given by Object-Z is the *class* construct which groups the definition of a state schema and the definitions of its associated operations. A class is a template for *objects* of that class: for each such object, its states are instances of the state schema of the class and its individual state transitions conform to individual operations of the class. An object is said to be an instance of a class and to evolve according to the definitions of its class. Syntactically, a class definition is a named box. In this box the constituents of the class are defined and related. The main constituents are: a state schema, an initial state schema and operation schemas. Consider the following specification of the class *Station* which denotes a store of messages of a given type [*Msg*]. The class contains an operation to send out a message to a *Car* class that contains an operation to receive a message.



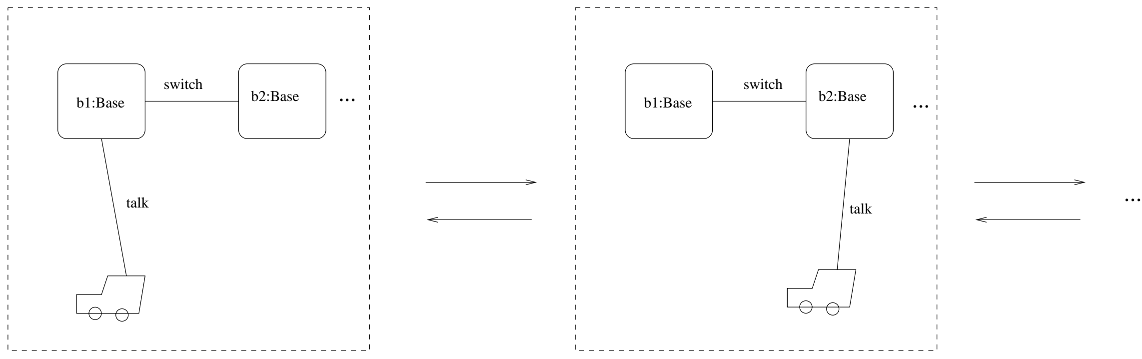
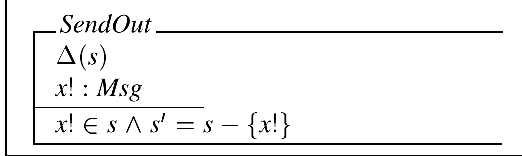
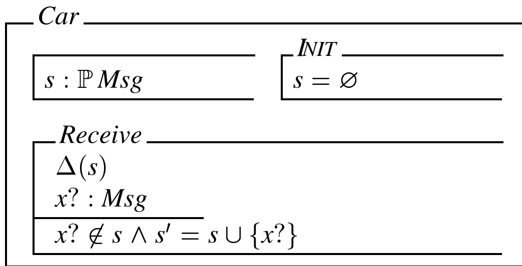


Figure 1. stations and a car





In this example, our classes *Station* and *Car* have one attribute s denoting a set of elements of the predefined type *Msg*. Operation schemas have a Δ-list of those attributes whose values may change. By convention, no Δ-list means no attribute changes value. Every operation schema implicitly includes the state schema in un-primed form s (the state before the operation) and primed form s' (the state after the operation).
In this example, operation *Receive* in *Car* class adds a given input $x?$ to the existing set (an identifier ending in ? denotes an input). Operation *SendOut* of *Station* class outputs a value $x!$ defined as one element of s and reduces s by deleting $x!$ from the original set (an identifier ending in ! denotes an output).
In Object-Z, the communication between objects can be modeled by the parallel operator $\|$. For example, if

$$st : Station;\ c : Car$$

then

$$st.SendOut \parallel c.Receive$$

captures that a message has been sent from a station st to a car c. The parallel operator $\|$ joins constraints and equates variables with the same name and also equates and hides any input variable to one of the components of $\|$ with any output from the other component that has the same name (i.e. the inputs and outputs are denoted by the same identifier apart from ? and ! decorations).

Difficulty with communication dynamics

Object-Z can be used to model fixed communication topologies and also can be effective for modeling any dynamic systems with its reference semantics. However our focus is to bring the power of π-calculus and Object-Z together without heavily reply on cross referencing. This motivates our work on embedding π-calculus constructs into Object-Z.

3. Linking π with Object-Z

Various modeling methods can be used in an effective combination for designing complex systems if the semantic links between those methods can be clearly established. The semantic/syntax integration of those methods would be a consequence of well defined semantic links. In PiOZ, the fundamental semantic links between π-calculus and Object-Z are:

- channels in π-calculus are semantically identified as state variables in Object-Z classes;
- processes and input guards in π-calculus are semantically identified to Object-Z operations which may perform state updates. For example,

$$\overline{c}\langle y\rangle;\ P \ \mid\ c(x);\ Q \ \longrightarrow$$

$$\boxed{\begin{array}{l}\Delta(x)\\ \hline x' = y\end{array}}\ ;\ (P \mid Q)$$

Furthermore, we enforce all communications must be through channels. Since all input/output are π-channel based, Z style input/output(?/!) parameters are replaced in PiOZ by parameters coming from the π-calculus syntax. Channels can be declared in two levels, class and operation. **chan**$[X]$ is the predefined type that contains all channels that carry messages of type X and a special value *nil* (i.e. $nil \in$ **chan**$[X]$) blocking any communication. For any declarations t_1, t_2 : **chan**$[X]$, t_1, t_2 are treated as state variables declarations, the same as other Object-Z state variables (i.e. it's possible that $t_1 \neq t_2$ or $t_1 = t_2$). Since channels variables can be specified in the class invariant or state guards so that the notion of *private channels* can be modelled.

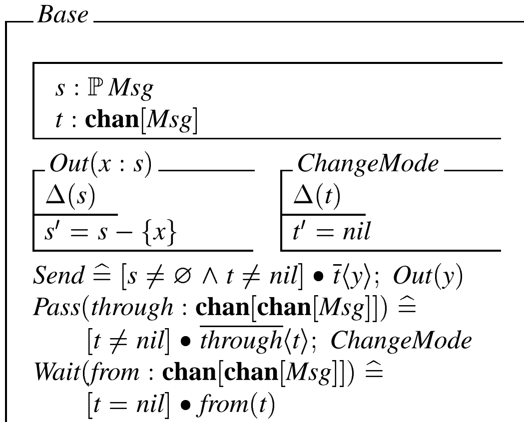
Now consider the following additional requirements to the previous stations-car example. Namely

- an arbitrary group of base stations (instead of just two bases) are involved.
- *talk* channel can be passed back and forth between bases (instead of just one switch).
- when a base station passes a message to the car, the base station should have one less message in its collection and the car should have one more message.
- the system has an active behaviour (either message passing between a station and the car or communication link switches between stations) if a base station has messages to send; it may terminate if all base stations do not have any messages (their "s" variable is the empty set).

A base station can

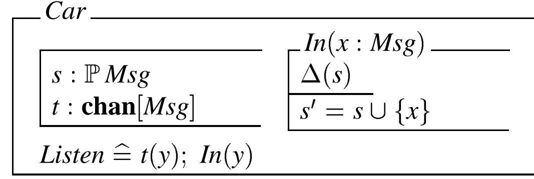
- send messages to the car if the car is in its region;
- pass the car communication channel to another base; and
- wait for the car communication channel from another base.

and it is specified by the following class

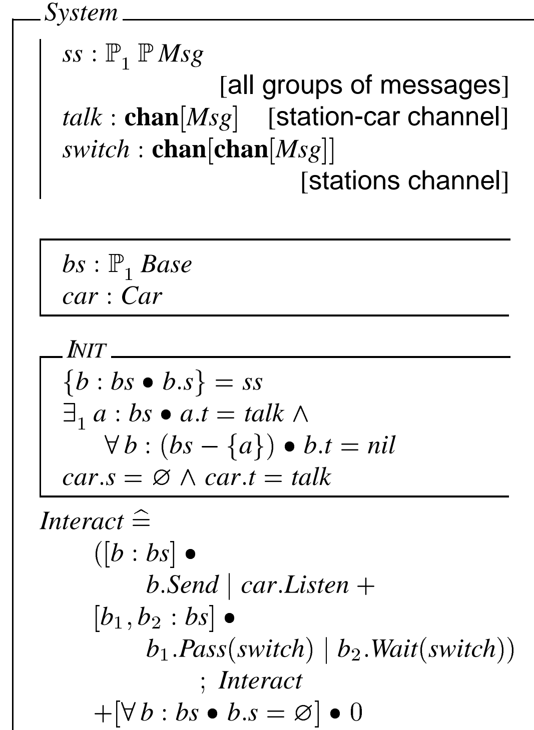


Note that variable t of the channel type can be included in the Δ list (as any other variable). The carrying type of **chan**$[Msg]$ can deduce the type of the variable 'y' in the operation *Send*.

The car in this case is to take input messages from channel t and record them (unbounded).



Now instances of *Base* and *Car* can be composed/linked together to form the *System* class:



The system active behaviour with possible termination is captured by the operation *Interact*.

System properties can be formulated. For example, no message loss can be stated as:

$$System \bullet car.s \cup \bigcup\{b : bs \bullet b.s\} \;=\; \bigcup ss$$

This property always holds before and after each *Interact* cycle. The induction proof is given in Section 6.

4. Syntax and Typing

The main PiOZ extension to Object-Z is to include π-style process definitions:

Class
OZDefinition
ProcessDefinition

A class has an Object-Z part *OZDefinition* which obeys the conventional syntactic definition [23] except newly introduced parametric operation. The original Object-Z operation operators: parallel composition, nondeterministic choice, sequential composition are replaced by π-process operators.
The process part *ProcessDefinition* is an extension of the syntax of the π-calculus which includes state guard and Object-Z operations specified in that class. We will use $a, b, \ldots$ for object references, $x, y, \ldots$ for channels, $v, u, v_1, v_2, \ldots$ for variables, where we assume that channels are included in variables, $Op(v_1, \ldots, v_n)$ for parametric operation, R for the process identifier which defines a process definition, G for any expression in Object-Z which serves as a guard, and $T_1, T_2, \ldots$ for types. It must be noted that only monadic channels are used in PiOZ.

$$\begin{aligned}\alpha \quad ::= \quad & \overline{x}\langle v \rangle \;\mid\; x(v) \;\mid\; Op(v_1, \ldots, v_n) \\ & \mid a.Op(v_1, \ldots, v_n)\end{aligned}$$

Processes are defined as

$$\begin{aligned}P ::= \quad & 0 \;\mid\; \alpha \;\mid\; P;\, Q \;\mid\; P \mid Q \;\mid\; P + Q \\ & \mid [G] \bullet P \;\mid\; R(x_1 : T_1, \ldots, x_n : T_n)\end{aligned}$$

The syntax of process definition is given by

$$R(x_1 : T_1, \ldots, x_n : T_n) \widehat{=} P$$

In order to avoid the confusion which might be caused by the dot notation used both as the sequential composition in the π-calculus and the object reference in Object-Z, we chose '; ' for the sequential operator in the process definition and reserve '.' for the object reference. *ProcessDefinition* in the above class is a collection of process definitions of the following form:

$$\begin{aligned}& R_1(x_{11} : T_{11}, \ldots, x_{1l} : T_{1l}) \widehat{=} P_1 \\ & \ldots \\ & R_n(x_{n1} : T_{n1}, \ldots, x_{nm} : T_{nm}) \widehat{=} P_n\end{aligned}$$

4.1. Type System

In this section we will present an integrated typing system for the PiOZ notation in which we will reconcile two different type systems of Object-Z and the π-calculus.

4.2. Typing Rules of Object-Z

OZDefinition will be type-checked by the conventional typing rules of Object-Z [23]. The main purpose of our type system is to ensure that communications between processes are well typed and reduction rules are well defined.
We are not going into details of Object-Z type system and assume that we have the following typing judgement relation for expressions in Object-Z:

$$\Gamma \vdash OZ_Expr$$

where Γ is a finite partial function from variables to types. Hence, we have the following typing rule for Object-Z expressions:

Typing Rules (Object-Z):

$$\frac{}{\Gamma \vdash OZ_Expr}$$

4.3. Type System for PiOZ

We will basically follow the typing system for process terms in the π-calculus given by Sewell [22].
The following judgement relation will be used for typing process definitions:

$$\Gamma \vdash \bar{t}\langle y \rangle;\; Out(y)\ \textbf{proc}$$

The above typing judgement reads "under Γ, $\bar{t}\langle y \rangle;\; Out(y)$ is well-typed". Any types which are defined in Object-Z can be used as types in PiOZ processes so that Object-Z specification part and process definition part share the same Γ.

Typing Rules (Values):

$$\frac{}{\Gamma, x : T \vdash x : T}$$

Typing Rules (Processes):

$$\frac{\Gamma \vdash x : \textbf{chan}\,[T] \quad \Gamma \vdash v : T \quad \Gamma \vdash P \quad \textbf{proc}}{\Gamma \vdash \overline{x}\langle v \rangle\ ;\ P \quad \textbf{proc}}$$

$$\frac{\Gamma \vdash x : \textbf{chan}\,[T] \quad \Gamma \vdash v : T \quad \Gamma \vdash P \quad \textbf{proc}}{\Gamma \vdash x(v)\ ;\ P \quad \textbf{proc}}$$

$$\frac{\Gamma \vdash P \quad \textbf{proc} \qquad \Gamma \vdash Q \quad \textbf{proc}}{\Gamma \vdash P \mid Q \quad \textbf{proc}}$$

$$\frac{\Gamma \vdash P \quad \textbf{proc} \qquad \Gamma \vdash Q \quad \textbf{proc}}{\Gamma \vdash P + Q \quad \textbf{proc}}$$

$$\frac{}{\Gamma \vdash 0 \quad \textbf{proc}}$$

Typing Rules (OZ Processes):
The last two rules allow *Op* not to have any parameter.

$$\frac{\Gamma \vdash G \qquad \Gamma \vdash P \ \ \textbf{proc}}{\Gamma \vdash [G] \bullet P \ \ \textbf{proc}}$$

$$\frac{\Gamma \vdash v_1 : T_1 \ldots \Gamma \vdash v_n : T_n \qquad \Gamma \vdash Q \ \ \textbf{proc}}{\Gamma \vdash R(v_1 : T_1, \ldots, v_n : T_n) \mathrel{\widehat{=}} Q \ \ \textbf{proc}}$$

$$\frac{\Gamma \vdash Op(x_1, \ldots, x_n)}{\Gamma \vdash Op(x_1 \ldots, x_n) \ \ \textbf{proc}}$$

$$\frac{\Gamma \vdash a.Op(x_1, \ldots, x_n)}{\Gamma \vdash a.Op(x_1, \ldots, x_n) \ \ \textbf{proc}}$$

5. Semantics and Derivation

In this section we present a formal description of the operational behaviour of PiOZ specifications. In this way we provide a dynamic description of the interaction and communication among processes by taking care of the static state-based description provided by OZ specification.

5.1. Operational Semantics

Starting from a basic formalization of the operational semantics of the π-calculus given by a labelled transition system (see [17]), we present an abstract machine $< ProcTerms \times States, \{\xrightarrow{\alpha} | \ \alpha \in Actions\} >$ where *ProcTerms* represents the set of all processes and process definitions, and *States* represents the set of all states. We work with configurations of form $< P, \sigma >$ where P is a process term and σ is a state. We have two kinds of transitions: one for the evolution of objects and another for their interaction. The transitions from a configuration to another configuration are described by a transition relation. For object evolution the transition relation is defined by $\rightarrow \subseteq (ProcTerms \times States) \times (ProcTerms \times States)$. For object interaction the transition relation is defined by $\rightarrow \subseteq (ProcTerms \times States) \times Actions \times (ProcTerms \times States)$. According to the case, we write either
$< P, \sigma > \longrightarrow < P', \sigma' >$ or $< P, \sigma > \stackrel{\alpha}{\longrightarrow} < P', \sigma' >$
where σ and σ' are semantic functions in *States* which represent the states before and after the object evolution.
α is an action defined by

$$\alpha \quad ::= \quad \overline{x} \ | \ x$$

We use the actions α to label the input and output guarded processes.
The reduction relation is defined by the structural congruence and derivation rules. The followings are structural congruences which will be used by PiOZ:

- $P + 0 \equiv P, P + Q \equiv Q + P, (P + Q) + R \equiv P + (Q + R)$,
- $P \mid 0 \equiv P, P \mid Q \equiv Q \mid P, (P \mid Q) \mid R \equiv P \mid (Q \mid R)$,
- $0; \ P \equiv P, P; \ 0 \equiv P$.

Before we start to define derivation rules, we need some definitions for the validity of Object-Z expressions. The fact that a state guard G is valid under the semantic function σ is denoted by the following notation:

$$\sigma \vDash G$$

which reads G is valid under the semantic function σ. The fact that an operation $Op(x_1, \ldots, x_n)$ is valid under the semantic function σ, σ' is denoted by

$$\sigma, \sigma' \vDash Op(x_1, \ldots, x_n)$$

We describe the operational behaviour of PiOZ by the derivation rules presented in Table 1. These rules are applied to systems of concurrent objects. Each object has its own evolution and from time to time two objects can interact along the channels they share. We don't consider concurrent processes running inside the same object. This means that we do not consider concurrent OOP. In some sense, we present an alternative to concurrent OOP, another way of including concurrency and mobility. We use the actions α to label the input and output guarded processes. Two rules are devoted to our guarded processes $\overline{x}\langle v \rangle; \ P$ and $x(v); \ P$.
Dynamic semantics derivations are related to execution. Modern programming languages (such as ML and Java) are "type safe"; this means that certain kinds of mismatches cannot arise during execution. Type safety is a relation between the static and dynamic semantics. Type safety provides information about the execution of well-typed programs. Our derivation rules consider only well-typed PiOZ expressions.
The communication between two objects is coming together with name/value passing. This name/value passing is described by the rule

$$\frac{\sigma, \sigma_1' \vDash a.Op_1 \quad \sigma, \sigma_2' \vDash b.Op_2}{< a.Op_1 \ | \ b.Op_2, \sigma > \rightarrow < 0, \sigma_1' \oplus (\sigma_2' - \sigma) >} \ a \neq b$$

where the states σ_1' and σ_2' represent the result of $a.Op_1$ and $b.Op_2$ respectively. The name/value passing is determined by the final state $\sigma_1' \oplus (\sigma_2' - \sigma)$. Note that there is no interference between $a.Op_1$ and $b.Op_2$ (simply because two different objects do not share common state). Therefore the final state is equivalent to $\sigma_2' \oplus (\sigma_1' - \sigma)$.
We consider an *object propagation*. This refer to the fact that, for instance, when we have *car.Listen*, then we apply our expansion rules and we get $t(y); \ car.In(y)$. We use the following rules to expand expressions associated with object references.

Expansion Rules:

$$
\begin{array}{rcl}
expand_a(G \bullet P) & = & [expand_a(G)] \bullet expand_a(P) \\
expand_a(P \mid Q) & = & expand_a(P) \mid expand_a(Q) \\
expand_a(P + Q) & = & expand_a(P) + expand_a(Q) \\
expand_a(P;\ Q) & = & expand_a(P);\ expand_a(Q) \\
expand_a(\overline{x}\langle v \rangle) & = & \overline{x}\langle v \rangle \\
expand_a(x(v)) & = & x(v) \\
expand_a(0) & = & 0 \\
expand_a(Op(v_1, \ldots, v_n)) & = & a.Op(v_1, \ldots, v_n)
\end{array}
$$

where $expand_a(G)$ will follow the rules of object reference in Object-Z.

6. Reasoning and Verification

In order to verify *invariant* and *liveness* properties we will adopt the verification procedure proposed by Evans [8]. His original idea is to verify those properties for concurrent Z specifications by using assertional methods proposed by Misra and Chandy [3]. PiOZ is an integration of the π-calculus and Object-Z so that we need to accommodate the method both in process algebraic and object-oriented settings.

Theorem 1 *(no message loss)*

$$car.s \cup \bigcup\{b : bs \bullet b.s\} = \bigcup ss$$

Proof: We need to prove the following cases:

$$INIT \vdash car.s \cup \bigcup\{b : bs \bullet b.s\} = \bigcup ss$$

and for each *Interact* circle,

$$
\begin{array}{l}
Interact \vdash car.s \cup \bigcup\{b : bs \bullet b.s\} = \bigcup ss \\
\quad \Rightarrow car.s' \cup \bigcup\{b : bs \bullet b.s'\} = \bigcup ss
\end{array}
$$

Note that ss is not primed, since it is defined as a global constant.
Any object specification in the Object-Z part is treated as logical formulas which are referenced in object reference notation in proofs.
[Initialization]

$$
\begin{array}{lll}
INIT & \vdash & \{b : bs \bullet b.s\} = ss \\
& \vdash & \bigcup\{b : bs \bullet b.s\} = \bigcup ss \\
& \vdash & \varnothing \cup \bigcup\{b : bs \bullet b.s\} = \bigcup ss \\
& \vdash & car.s \cup \bigcup\{b : bs \bullet b.s\} = \bigcup ss
\end{array}
$$

[Operations]

Basically, operation *Interact* has three branches:

1. $[b : bs] \bullet b.Send \mid car.Listen$
2. $[b_1, b_2 : bs] \bullet b_1.Pass(switch) \mid b_2.Wait(switch)$
3. $[\forall\, b : bs \bullet b.s = \varnothing] \bullet 0$

For case 1, by using the expansion rules for expressions, we have

$$
\begin{array}{l}
[b : bs] \bullet b.Send \mid car.Listen \Leftrightarrow \\
[b : bs \wedge b.s \neq \varnothing \wedge t \neq nil] \bullet \\
\qquad \overline{t}\langle y \rangle;\ b.Out(y) \mid t(y);\ car.In(y) \Leftrightarrow \\
b : bs \wedge y : Msg \wedge b.s \neq \varnothing \wedge t \neq nil \vdash \\
\qquad b.s' = b.s - \{y\} \wedge car.s' = car.s \cup \{y\}
\end{array}
$$

where we have used the following expansions of operation schemas

$$
\begin{array}{l}
b.Out(y) \equiv b.s' = b.s - \{y\} \\
car.In(y) \equiv car.s' = car.s \cup \{y\}
\end{array}
$$

and two basic principles such that all guards are combined as hypotheses, and that input variables of input channel and output value of its complementary output channel in communication is regarded as equation.
Based on the following lemma:

Lemma 1

1. $b : bs \vdash \bigcup\{bb : bs \bullet bb.s\} = \bigcup\{bb : bs - \{b\} \bullet bb.s\} \cup b.s$
2. $b : bs \vdash \{bb : bs - \{b\} \bullet bb.s'\} = \{bb : bs - \{b\} \bullet bb.s\}$

we can deduce

$$
\begin{array}{l}
car.s' \cup \bigcup\{bb : bs \bullet bb.s'\} = \\
car.s \cup \{y\} \cup \bigcup\{bb : bs \bullet bb.s'\} = \\
car.s \cup \{y\} \cup \bigcup\{bb : bs - \{b\} \bullet bb.s'\} \cup b.s' = \\
car.s \cup \{y\} \cup \bigcup\{bb : bs - \{b\} \bullet bb.s'\} \cup b.s - \{y\} = \\
car.s \cup \{y\} \cup \bigcup\{bb : bs - \{b\} \bullet bb.s\} \cup b.s - \{y\} = \\
car.s \cup \bigcup\{bb : bs - \{b\} \bullet bb.s\} \cup b.s = \\
car.s \cup \bigcup\{bb : bs \bullet bb.s\} = \bigcup ss
\end{array}
$$

For cases 2 and 3, the property hold obviously since there is no state updates on message variables. The property is proved, by induction.

7. Related Work and Conclusion

Methods integration has become a recent research trend in software specification and design. In graphical area, many object-oriented methods merged into one, the UML. In formal symbolic area, methods integration is even more motivated as many traditional formal methods do not scale-up well. Formal methods integration is a recent research focus [1, 14, 2]. Much methods integration efforts are concentrated on blending Z with either CSP [9, 16, 26, 24, 30] or CCS [12, 27]. However, the notion of dynamic communication configuration (mobility) of π-calculus offers significant

$$(R_1)\quad \frac{< expand_{\sigma(var)}(P\{v_1/x_1,\ldots,v_n/x_n\}),\sigma > \longrightarrow < Q,\sigma' >}{< var.R(v_1,\ldots,v_n),\sigma > \longrightarrow < Q,\sigma' >}\; R(x_1:T_1,\ldots,x_n:T_n) \mathrel{\widehat{=}} P$$

$$(R_2)\quad \frac{\sigma,\sigma' \models Op(v_1,\ldots,v_n)}{< Op(x_1,\ldots,x_n)\ ;\ P,\sigma > \longrightarrow < P,\sigma' >} \qquad (R_3)\quad \frac{< P,\sigma > \longrightarrow < P',\sigma' >}{< P \mid Q,\sigma > \longrightarrow < P' \mid Q,\sigma' >}$$

$$(R_4)\quad \frac{\sigma,\sigma'_1 \models a.Op_1(x_1,\ldots,x_n) \quad \sigma,\sigma'_2 \models b.Op_2(y_1,\ldots,y_m)}{< a.Op_1(x_1,\ldots,x_n) \mid b.Op_2(y_1,\ldots,y_m),\sigma > \longrightarrow < 0,\sigma'_1 \oplus (\sigma'_2 - \sigma) >}\; a \neq b$$

$$(R_5)\quad \frac{< \overline{x}\langle v\rangle;\ P,\sigma > \xrightarrow{\overline{x}} < P,\sigma > \quad < w(u);\ Q,\sigma > \xrightarrow{w} < Q,\sigma >}{< \overline{x}\langle v\rangle;\ P \mid w(u);\ Q,\sigma > \longrightarrow < (P \mid Q),\sigma \oplus \{(u,\sigma(v))\} >}\; \sigma(x) = \sigma(w) \neq nil$$

$$(R_6)\quad \frac{\sigma \models pred}{< [pred] \bullet P,\sigma > \longrightarrow < P,\sigma >} \qquad (R_7)\quad \frac{\sigma \models [x:T \mid p(x)] \bullet P}{< [x:T \mid p(x)] \bullet P,\sigma > \longrightarrow < P,\sigma \oplus \{(x,\mu\, v \mid p(v))\} >}$$

$$(R_8)\quad \frac{< P,\sigma > \longrightarrow < P',\sigma' >}{< P;\ Q,\sigma > \longrightarrow < P';\ Q,\sigma' >} \qquad (R_9)\quad \frac{< P,\sigma > \longrightarrow < P',\sigma' >}{< P + Q,\sigma > \longrightarrow < P',\sigma' >} \qquad (R_{10})\quad \frac{< P,\sigma > \longrightarrow < P',\sigma' > \quad P' \equiv P''}{< P,\sigma > \longrightarrow < P'',\sigma' >}$$

Table 1. Derivation rules of PiOZ

power and extra dimensions if it can be incorporated with a state-based formalism. An attempt to define a machine-like (state-based) formalism for the π-calculus was presented in [4]. Incorporating the dynamic communication configuration of π-calculus with a state-based formalism is the central issue which this paper tries to resolve.

PiOZ builds on ideas from our earlier work in [16, 27]. For instance, the work in [16] influences our approach to treat the Object-Z operation and π-calculus process as the same semantic and syntactic entity in PiOZ. The new ideas of this work is to use channel-typed state *variables* to define the communication interface of PiOZ objects. In particular, the π-calculus input guard (binding operator) is given a state-update semantics. As a result, the formal operational semantics of PiOZ integrates effectively the Object-Z state transition semantics and π-calculus reduction rules, and provides a foundation for reasoning about system properties in PiOZ. The Z style of input/output(?/!) are completely replaced by channel input/output indicators with parameterised processes. PiOZ not only extends Object-Z with advanced π-style dynamic communicating capabilities, but also supports π-style instance level reductions (derivation/animation). The typing rules of PiOZ has been constructed based on π-calculus and Object-Z typing rules. The major difference of PiOZ to most state/event methods integration is obvious: π-style dynamic communicating configuration is supported by PiOZ, while others followed the CSP-style of fixed communicating configuration. Note that our recent work on mobile Object-Z [28] is similar to the Mobile Unity's work [11] (an extension to Unity [3]). Mobile Object-Z and mobile unity focus on locality. Mobile Object-Z has built the mobile primitives such as "go(Address)" and "here(Address)" within the Object-Z notation, which is very different from the approach presented in this paper — the focus of PiOZ is on embedding π-calculus into Object-Z to support dynamic communication configurations.

One of the areas of future work will be on tools support. PiOZ preserves in large part both the syntax and semantics of Object-Z and π-calculus and hence may potentially benefit from the body of experience developed in the use of and tool support for the individual notations. Our operational semantics with π-style instance level reductions may provide useful guidelines for developing a PiOZ analysis/animation tool. We have developed the XML environment for PiOZ and currently investigating the checking and visualization tools for PiOZ based on our recent Object-Z type checking and visualisation tool [6] and the various π-calculus checking tools [5, 20, 29]. Recent research work on encoding of π-calculus and Object-Z into Isabelle/HOL [21, 25] will also provide useful foundations for the future embedding PiOZ into a theorem prover.

Acknowledgements

This work is supported by the academic research grant *Integrated Formal Methods* (R-252-000-050-107) from Na-

tional University of Singapore.

References

[1] K. Araki, A. Galloway, and K. Taguchi, editors. *IFM'99: Integrated Formal Methods, York, UK*. Springer-Verlag, June 1999.

[2] M. Butler, L. Petre, and K. Sere, editors. *IFM'02: Integrated Formal Methods,*, volume 2335 of *Springer. Lecture Notes in Computer Science*. Springer-Verlag, January 2002.

[3] K. M. Chandy and J. Misra. *Parallel Programming Design*. Addison-Wesley, 1988.

[4] G. Ciobanu and M. Rotaru. A π-calculus machine. *Journal of Universal Computer Science*, 6 (1):39–59, 2000.

[5] M. Dam. Model checking mobile processes. *Journal of Information and Computation*, 129(1):35–51, 1996.

[6] J. S. Dong, Y. F. Li, J. Sun, J. Sun, and H. Wang. XML-based static type checking and dynamic visualization for TCOZ. In George and Miao [13], pages 311–322.

[7] R. Duke and G. Rose. *Formal Object Oriented Specification Using Object-Z*. Cornerstones of Computing. Macmillan, March 2000.

[8] A. Evans. Specifying & verifying concurrent systems using Z. In *Proceedings of FME'94: Industrial Benefit of Formal Methods*, pages 366–400. Springer-Verlag, 1994.

[9] C. Fischer and H. Wehrheim. Model-Checking CSP-OZ Specifications with FDR. In Araki et al. [1], pages 315–334.

[10] J. Fischer, A. Prinz, and A. Vogel. Different FDT's confronted with different ODP-viewpoints of the trader. In J. C. P. Woodcock and P. G. Larsen, editors, *FME'93: Industrial Strength Formal Methods*, volume 670 of *Springer. Lecture Notes in Computer Science*, pages 332–350. Springer-Verlag, April 1993.

[11] P. J. McCann G.-C. Roman and J. Y. Plun. Mobile unity: reasoning and specification in mobile computing. *ACM Trans. Software Engineering and Methodology*, 6(3):250–282, 1997.

[12] A. J. Galloway and W. J. Stoddart. An operational semantics for ZCCS. In Hinchey and Liu [15], pages 272–282.

[13] C. George and H. Miao, editors. *International Conference on Formal Engineering Methods (ICFEM'02)*, volume 2495 of *Springer. Lecture Notes in Computer Science*. Springer-Verlag, October 2002.

[14] W. Grieskamp, T. Santen, and B. Stoddart, editors. *IFM'00: Integrated Formal Methods,*, volume 1945 of *Springer. Lecture Notes in Computer Science*. Springer-Verlag, October 2000.

[15] M. Hinchey and S. Liu, editors. *the IEEE International Conference on Formal Engineering Methods (ICFEM'97)*, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

[16] B. Mahony and J.S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177, February 2000.

[17] R. Milner. *Communicating and mobile systems : the π-calculus*. Cambridge University Press, 1999.

[18] R. Milner, J. Parrow, and J. Walker. A Calculus of Mobile Processes Part I and II. *Information and Computation*, 100:1–77, 1992.

[19] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirement specifications. *IEEE Trans. Software Eng.*, 20(10):760–773, October 1994.

[20] S. K. Rajamani and J. Rehof. A behavioral module system for the pi-calculus. In *Proceedings SAS 01, Static Analysis Symposium, Paris, France*, 2001.

[21] C. Röckl, D. Hirschkoff, and S. Berghofer. Higher-order abstract syntax with induction in Isabelle/HOL: Formalizing the pi-calculus and mechanizing the theory of contexts. In F. Honsell and M. Miculan, editors, *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'01), Held as Part of the Joint ETAPS'01, (Genova, Italy, April 2001)*, volume 2030 of *Springer. Lecture Notes in Computer Science*, pages 364–378. Springer, 2001.

[22] P. Sewell. Applied pi - a brief tutorial. Technical report 498, Computer Laboratory, University of Cambridge, 2000.

[23] G. Smith. *The Object-Z Specification Language*. Advances in Formal Methods. Kluwer Academic Publishers, 2000.

[24] G. Smith and J. Derrick. Specification, refinement and verification of concurrent systems - an integration of Object-Z and CSP. *Formal Methods in System Design*, 18:249–284, 2001.

[25] G. Smith, F. Kammller, and T. Santen. Encoding object-z in isabelle/hol. In *2nd International Conference of Z and B Users (ZB'02)*, volume 2272 of *Springer. Lecture Notes in Computer Science*, pages 82–99. Springer, 2002.

[26] C. Suhl. RT-Z: An integration of Z and timed CSP. In Araki et al. [1], pages 29–48.

[27] K. Taguchi and K. Araki. The State-Based CCS Semantics for Concurrent Z Specification. In Hinchey and Liu [15], pages 283–292.

[28] K. Taguchi and J. S. Dong. An Overview of Mobile Object-Z. In George and Miao [13], pages 144–155.

[29] B. Victor and F. Moller. The Mobility Workbench — A tool for the π-calculus. In D. Dill, editor, *Computer Aided Verification (Proc. of CAV'94)*, volume 818 of *Springer. Lecture Notes in Computer Science*, pages 428–440. Springer, 1994.

[30] J. Woodcock and A. Cavalcanti. The steam boiler in a unified theory of Z and CSP. In *The 8th Asia-Pacific Software Engineering Conference (APSEC'01)*, pages 291–298. IEEE Press, 2001.

[31] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall International, 1996.