Distributed Semantics and Implementation for Systems with Interaction and Priority

Ananda Basu, Philippe Bidinger, Marius Bozga, and Joseph Sifakis

Université Grenoble 1 - CNRS - VERIMAG
Centre Équation, 2 av de Vignate, 38610 Gières, France

Abstract. The paper studies a distributed implementation method for the BIP (Behavior, Interaction, Priority) component framework for modeling heterogeneous systems.

BIP offers two powerful mechanisms for describing composition of components by combining interactions and priorities. A system model is layered. The lowest layer contains atomic components; the second layer, describes possible interactions between atomic components; the third layer includes priorities between the interactions. The current implementation of BIP is based on global state operational semantics. An Engine directly interprets the operational semantics rules and computes the possible interactions between atomic components from global states.

The implementation method is a translation from BIP models into distributed models involving two steps. The first translates BIP models into partial state models where are known only the states of the components which are ready to communicate. The second implements interactions in the partial state model by using message passing primitives.

The main results of the paper are conditions for which the three models are observationally equivalent. We show that in general, the translation from global state to partial state models does not preserve observational equivalence. Preservation can be achieved by strengthening the premises of the operational semantics rules by an oracle. This is a predicate depending on the priorities of the BIP model. We show that there are many possible choices for oracles. Maximal parallelism is achieved for dynamic oracles allowing interaction as soon as possible. Nonetheless, these oracles may entail considerable computational overhead. We study performance trade-offs for different types of oracles. Finally, we provide experimental results illustrating the application of the theory on a prototype implementation.

1 Introduction

A distributed system is a collection of loosely coupled independent components, communicating by explicit message passing. The components are intrinsically concurrent and their states may be known only through communication. We cannot determine the exact global state of a distributed system, we can only approximate it [4].

K. Suzuki et al. (Eds.): FORTE 2008, LNCS 5048, pp. 116–133, 2008.

The paper studies a distributed implementation method for the BIP (Behavior, Interaction, Priority) component framework for modeling heterogeneous systems [2]. The method consists of three steps:

• It starts from a *global state* model of the system to be implemented described in BIP. The model represents the system behavior as a transition system where transitions are atomic. The BIP execution platform uses an Engine which coordinates the execution of the components. Atomicity of transitions implies a strict alternation between the execution of components and the Engine: no interaction is possible when some component is performing a computation.

• From the global state model, a *partial state* model is derived where we distinguish between states from which components are *ready* for interaction and states where components are *busy* by executing some internal computation. For this model partial state knowledge may suffice for executing interactions. We study conditions for the partial state model to be equivalent to the global state model. The conditions are in the form of an *oracle* used by the BIP Engine to safely execute interactions in the presence of uncertainty about the global state.

• From the partial state model, a *distributed model* is obtained where atomic multiparty interactions of the partial state models are replaced by communication protocols. In this model, components exchange messages to communicate with the Engine represented by an additional component.

The main results of the paper are conditions for which the three models are observationally equivalent by considering as silent the actions corresponding to internal computations of the initial global state model. They are described in more details below.

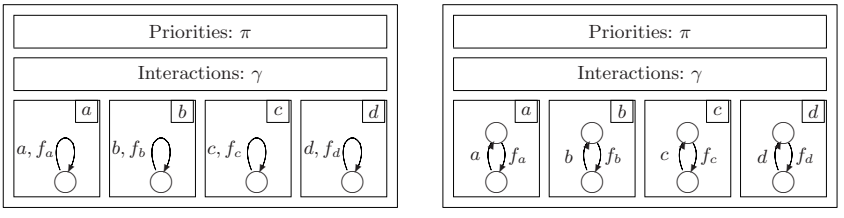
BIP combines two powerful mechanisms for describing multiparty interactions between components: *interactions* and *priorities*. A system model is layered. The lowest layer contains atomic components whose behavior is described by state machines with data and functions described in C. As in process algebras, atomic components can communicate by using ports. The second layer contains interactions which are relations between communication ports of individual components. Priorities are used to express scheduling policies by selecting amongst the enabled interactions of the layer underneath.

The current implementation of BIP is based on global state semantics. From a BIP model, a compiler is used to generate C++ code for a dedicated platform. The platform uses an Engine that directly interprets the operational semantics rules. For a given global state, the Engine computes from the set of the communication ports offered by individual components and the set of interactions, the set of the enabled interactions. Amongst these, the Engine chooses a maximal one, according to the priorities of the third layer, and notifies the involved components which can continue their computation.

We define partial state semantics for BIP where the assumption of atomic execution of transitions does not hold. This is a straightforward generalization of global state semantics where interactions are separated from internal computation in the components. A component may be either in a busy state or in a ready state. A busy state corresponds to the execution of some internal

computation. When the computation terminates, some ready state is reached. From this state the component can participate in interactions and move again to some busy state.

The implementation problem for a partial state model is to find an Engine that may execute interactions even for partially known states, while preserving (observational) equivalence with the corresponding global state model. The following example shows that in general, the two models are not equivalent.



(a) Global State Model (b) Partial State Model

Fig. 1. A System with Four Components

Example 1. Consider a BIP model consisting of four components A, B, C, D each one offering cyclically an interaction through ports a, b, c, d followed respectively by the execution of functions f_a, f_b, f_c, f_d (Figure 1(a)). We assume that A is a sender and B, C, D are receivers. A can broadcast a message through a and the set of the possible interactions is $\gamma = \{a, ab, ac, ad, abc, abd, acd, abcd\}$. Priority rules are used to ensure that amongst all the possible interactions from a state only a maximal one is possible. This is expressed by using a priority order on interactions π and rules of the form $x\pi xy$ where x and xy are interactions. These rules say that whenever both interactions x and xy are enabled, only interaction xy can be executed. That is, maximal progress is enforced. For this example, the only possible interaction is $abcd$ and thus the functions f_a, f_b, f_c, f_d are executed synchronously.

The partial state model for this system is shown in Figure 1(b). It is possible, due to the separation between interaction and internal computation, to reach a configuration where the receivers are in a busy state. In that case, only the ready components will be synchronized. Thus an arbitrary desynchronization of the receivers with respect to the sender is possible.

Example 2. Consider again the previous example where broadcast is replaced by three rendezvous: $\gamma = \{ab, bc, cd\}$ and π is such that $ab\pi bc,\ cd\pi bc$ in the global state system. This system executes forever the interaction bc. Consider the corresponding partial state system where interactions are separated from functions. For this system, it is possible to execute the sequence $ab.(f_a.cd.f_c.f_b.ab.f_d)^\omega$ which goes through states never enabling the interaction bc.

The above examples motivate the definition of partial state semantics where the premises of the operational semantics rules include an oracle, a predicate

parameterized by a dependency relation between interactions. The dependency relation is an abstraction of the priorities of the initial BIP model. The oracle characterizes the partial states from which an interaction can be safely executed: if an interaction a_1 depends on an interaction a_2, then a_1 cannot be executed if the system has some internal evolution leading to a state enabling a_2. We show that there are many possible choices for oracles. If the time for computing them is negligible, best performance is achieved for oracles allowing interaction as soon as possible in order to reduce waiting times of ready components. The worst performing oracle is the one allowing interaction only when all the components are at ready states. For this oracle partial and global state semantics coincide.

We study a transformation from the partial state model to a distributed one. This consists in replacing atomic interactions by protocols using message passing. For distributed semantics, the Engine becomes an additional component. The results are applied to obtain a multi-threaded implementation for BIP. We analyze performance of this implementation for different types of oracles as well as with respect to the global state semantics model.

The presented method is not specific to BIP and can be applied for the implementation of systems in particular in two cases. First, for concurrent systems with fairness constraints which at implementation level, become scheduling policies expressed by dynamic priorities. Second, for systems involving communication by broadcast. This requires mechanisms for identifying the maximal set of interacting components that can be specified by using priorities. Consequently, the proposed method can be used for correct implementation.

The paper is organized as follows. In section 2, we present global state semantics and the associated partial state semantics for BIP. In section 3, we study oracles and their properties. We show correctness of partial state semantics enforced by an oracle with respect to global state semantics. In section 4, we study the transformation from partial state to distributed semantics. We also discuss experimental results for a multi-threaded implementation, in particular for different choices of oracles. The last section includes conclusions and description of future work. Proofs are omitted due to space limitation but appear in [1].

2 BIP – Basic Semantic Models

2.1 Global State Semantics

BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority.

Atomic Components. We define *atomic components* as transition systems with a set of ports labeling individual transitions. These ports are used for communication between different components.

Definition 1 (Atomic Component). *An* atomic component B *is a labeled transition system represented by a triple* $(Q, P, \rightarrow)$ *where* Q *is a set of* states*,* P *is a set of* communication ports*,* $\rightarrow \subseteq Q \times P \times Q$ *is a set of* possible transitions*, each labeled by some port.*

For any pair of states $q, q' \in Q$ and a port $p \in P$, we write $q \xrightarrow{p} q'$, iff $(q, p, q') \in \rightarrow$. When the communication port is irrelevant, we simply write $q \rightarrow q'$. Similarly, $q \xrightarrow{p}$ means that there exists $q' \in Q$ such that $q \xrightarrow{p} q'$.

Interaction For a given system built from a set of n atomic components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1}^n$, we assume that their respective sets of ports are pairwise disjoint, *i.e.* for any two $i \neq j$ from $\{1..n\}$ we have $P_i \cap P_j = \emptyset$. We can therefore define the set $P = \bigcup_{i=1}^n P_i$ of all ports in the system. An *interaction* is a set $a \subseteq P$ of ports. When we write $a = \{p_i\}_{i \in I}$, we suppose that for $i \in I$, $p_i \in P_i$.

Definition 2 (Composite Component). *A* composite component *(or simply* component*) is defined by a composition operator parameterized by a set of interactions $\gamma \subseteq 2^P$. $B \stackrel{def}{=} \gamma(B_1, \ldots, B_n)$, is a transition system $(Q, \gamma, \rightarrow)$, where $Q = \bigotimes_{i=1}^n Q_i$ and $\rightarrow$ is the least set of transitions satisfying the rule*

$$\frac{a = \{p_i\}_{i \in I} \in \gamma \qquad \forall i \in I.\ q_i \xrightarrow{p_i}_i q_i' \qquad \forall i \notin I.\ q_i = q_i'}{(q_1, \ldots, q_n) \xrightarrow{a} (q_1', \ldots, q_n')}$$

The inference rule says that a composite component $B = \gamma(B_1, \ldots, B_n)$ can execute an interaction $a \in \gamma$, iff for each port $p_i \in a$, the corresponding atomic component B_i can execute the transition labeled with p_i; the states of components that do not participate in the interaction stay unchanged.

Observe that, it is possible for a composite component to further communicate on the ports initially provided by its atomic components

Priorities In composite components, many interactions can be enabled at the same time, introducing a degree of non-determinism in the product behavior. Non-determinism can be restricted by means of priorities, specifying which of the interactions should be preferred among the enabled ones.

Definition 3 (Priority Model). *A* priority *on $B = \gamma(B_1, \ldots, B_n)$ is a relation $\pi \subseteq \gamma \times Q \times \gamma$. We write $a_1 \pi_q a_2$ for $(a_1, q, a_2) \in \pi$. Furthermore, we require that for all $q \in Q$, π_q is a strict partial order on γ. $a_1 \pi_q a_2$ means that interaction a_1 has less priority than a_2 at state q.*

Given a behavior $B = (Q, P, \rightarrow)$ defined as above, we construct a new behavior $\pi B = (Q, P, \rightarrow_\pi)$ as follows:

$$\frac{q \xrightarrow{a} q' \qquad \forall a' \in \gamma.\ a \pi_q a' \implies q \stackrel{a'}{\not\rightarrow}}{q \xrightarrow{a}_\pi q'}$$

Example 3. The examples 1 and 2 are straightforward to define in BIP. The system Figure 1(a) is defined as $\pi\gamma(A, B, C, D)$ where A, B, C and D are atomic components with one state and one transition defined as $X = (\{q_X\}, \{x\}, (q_X, x, q_X))$ for $(X, x) \in \{(A, a), (B, b), (C, c), (D, d)\}$.

We have $\gamma = \{a, ab, ac, ad, abc, abd, abcd\}$. The system $\gamma(A, B, C, D)$ has only one state $q = (q_A, q_B, q_C, q_D)$ for which $\pi_q = \{(x, xy) \mid (x, xy) \in \gamma^2\}$. Example 2 is defined similarly for $\gamma = \{ab, bc, cd\}$ and $\pi_q = \{(ab, bc), (cd, bc)\}$.

Implementation The operational semantics rules are interpreted by the BIP *Engine*. At a given global state, each atomic component publishes the ports of the enabled transitions. From this information, the Engine computes the set of the possible interactions, that is the interactions of γ such that each one of their ports is published by some component. Amongst these interactions, the Engine chooses non-deterministically one that satisfies the priority rules π and notifies the involved components by communicating the corresponding port names.

2.2 Partial State Semantics

The model with global state semantics is based on the fact that transitions are atomic and a global state is always defined. To obtain the partial state model corresponding to a global state model, we 1) replace atomic components by their partial state models; 2) extend the operational semantics rules for interactions and priorities.

Atomic Components To model concurrent behavior, we associate with each atomic component, its corresponding *partial state model*. Atomic components with partial states behave as atomic components with the difference that each transition is decomposed into a sequence of two transitions: an interaction (visible transition) followed by an internal computation or *busy transition*. Between these two transitions, a new *busy* state is added. Busy states are transient states considered by the Engine as undefined states of the component.

Definition 4 (Atomic Component with Partial States). *Given an atomic component $B = (Q, P, \rightarrow)$, we define the associated partial state model as the transition system $B^{\perp} = (Q \cup Q^{\perp}, P \cup \{\beta\}, \rightsquigarrow)$ where*

- *$Q^{\perp} = \{q_t \mid t \in \rightarrow\}$ such that $Q^{\perp} \cap Q = \emptyset$. $Q^{\perp}$ is a set of busy states in bijection with the set of transitions $\rightarrow$.*
- *β is a port name not in P*
- *$\rightsquigarrow \subseteq (Q \cup Q^{\perp}) \times P \cup \{\beta\} \times (Q \cup Q^{\perp})$ where if $t = (q_1, p, q_2) \in \rightarrow$, then $q_1 \overset{p}{\rightsquigarrow} q_t$ and $q_t \overset{\beta}{\rightsquigarrow} q_2$.*

Interaction We define below interactions for partial state models.

Definition 5. *Given a BIP model built from a set of atomic components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1}^n$, of the form $\gamma(B_1, \ldots, B_n)$, we define the corresponding partial state model $\gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$ such that*

- *$B_i^{\perp}$ is the partial state model $B_i^{\perp} = (Q_i \cup Q_i^{\perp}, P_i \cup \{\beta_i\}, \rightsquigarrow_i)$*
- *$\gamma^{\perp} = \gamma \cup \{\beta_i\}_{i=1}^n$*

Notice that $\gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp}) = (\bigotimes_{i=1}^n (Q_i \cup Q_i^{\perp}), \gamma^{\perp}, \rightsquigarrow)$. The transition relation $\rightsquigarrow$ can be equivalently defined by the rules:

$$\frac{a = \{p_i\}_{i \in I} \in \gamma \qquad \forall i \in I.\ q_i \overset{p_i}{\rightsquigarrow}_i q_i' \qquad \forall i \notin I.\ q_i = q_i'}{(q_1, \ldots, q_n) \overset{a}{\rightsquigarrow} (q_1', \ldots, q_n')}$$

$$\frac{q_i \overset{\beta_i}{\rightsquigarrow} q_i'}{(q_1, \ldots, q_i, \ldots, q_n) \overset{\beta_i}{\rightsquigarrow} (q_1, \ldots, q_i', \ldots, q_n)}$$

The first rule is the same as the composition rule for the global state semantics. The second rule defines the busy transitions of the composite system.

The state space can be split into two disjoint sets $\bigotimes_{i=1}^{n}(Q_i \cup Q_i^{\perp}) = Q^g \cup Q^p$. The set of *global states* $Q^g = \bigotimes_{i=1}^{n} Q_i$ which is the set of states of $\gamma(B_1, \ldots, B_n)$. The set of *partial states* Q^p where at least one component is busy.

Definition 6. *For $q, q' \in Q^p \cup Q^g$, we write $q \overset{\beta}{\rightsquigarrow} q'$ if $q \overset{\beta_i}{\rightsquigarrow} q'$ for some i.*

Property 1. The relation $\overset{\beta}{\rightsquigarrow}$ is terminating and confluent. Thus, from any partial state, a unique global state is eventually reached by executing β-transitions.

Priority The above property is used to define priorities for partial state models. The priority relation at some partial state should agree with the priority relation at the global state reached by executing β-transitions.

Definition 7. *Given a BIP model $\pi\gamma(B_1, \ldots, B_n)$, the corresponding partial state model is $\pi^{\perp}\gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$ where $\pi^{\perp} \subseteq \gamma \times (Q^g \cup Q^p) \times \gamma$ such that $a_1 \pi_q^{\perp} a_2$ if $\exists q' \in Q^g$. $q \overset{\beta^*}{\rightsquigarrow} q' \wedge a_1 \pi_{q'} a_2$.*

Note that $\pi^{\perp}$ is a priority and it coincides with π on Q^g.

Example 4. The partial state model for Example 3 has the atomic components $A^{\perp}$, $B^{\perp}$, $C^{\perp}$ and $D^{\perp}$ with two states and two transitions defined by

$$X^{\perp} = (\{q_X, q_X^{\perp}\}, \{x, \beta_X\}, \{(q_X, x, q_X^{\perp}), (q_X^{\perp}, \beta_X, q_X)\})$$

where $(X, x) \in \{(A, a), (B, b), (C, c), (D, d)\}$. For the first system, $\gamma^{\perp} = \{a, ab, ac, ad, abc, abd, abcd\} \cup \{\beta_A, \beta_B, \beta_C, \beta_D\}$ and $\pi^{\perp}$ is such that for all states q in $\gamma^{\perp}(A^{\perp}, B^{\perp}, C^{\perp}, D^{\perp})$, we have $\pi_q^{\perp} = \{(x, xy) \mid (x, xy) \in \gamma^2\}$. For the second system, we have $\gamma^{\perp} = \{ab, bc, cd\} \cup \{\beta_A, \beta_B, \beta_C, \beta_D\}$ and $\pi^{\perp}$ is such that for all states q in $\gamma^{\perp}(A^{\perp}, B^{\perp}, C^{\perp}, D^{\perp})$, we have $\pi_q^{\perp} = \{(ab, bc), (cd, bc)\}$.

2.3 Comparing Global and Partial State Semantics

We study sufficient conditions for partial state models to be behaviorally equivalent to global state models. We use observational equivalence [8] for this comparison by considering that β-transitions are not observable. As noticed in the introduction (Example 1), observational equivalence is not preserved. The systems $\pi\gamma(A, B, C, D)$ and $\pi^{\perp}\gamma^{\perp}(A^{\perp}, B^{\perp}, C^{\perp}, D^{\perp})$ are not observationally equivalent. The global state model can perform only the maximal interaction *abcd* while in the partial state model, non maximal synchronization is possible. For instance, we have the transitions:

$$(q_A, q_B, q_C, q_D) \overset{abcd}{\rightsquigarrow} (q_A^{\perp}, q_B^{\perp}, q_C^{\perp}, q_D^{\perp}) \overset{\beta}{\rightsquigarrow} (q_A, q_B^{\perp}, q_C^{\perp}, q_D^{\perp}) \overset{a}{\rightsquigarrow} (q_A^{\perp}, q_B^{\perp}, q_C^{\perp}, q_D^{\perp})$$

Thus, in general, a BIP model is not observationally equivalent to its partial state model. Nonetheless, the following theorem shows that if $\pi = \emptyset$, $\gamma(B_1, \ldots, B_n)$ and $\gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$ are observationally equivalent.

We define observational equivalence of two transition systems $A = (Q_A, L \cup \{\beta\}, \rightarrow_A)$ and $B = (Q_B, L \cup \{\beta\}, \rightarrow_B)$. It is based on the usual definition of weak bisimilarity where β-transitions are considered unobservable.

Definition 8 (Weak Simulation). *A weak simulation over A and B is a relation $R \subseteq Q_A \times Q_B$ such that we have $\forall (q,r) \in R,\ a \in L.\ q \xrightarrow{a}_A q' \implies \exists r'.\ (q',r') \in R \wedge r \overset{\beta^* a \beta^*}{\rightarrow}_B r'$ and $\forall (q,r) \in R.\ q \xrightarrow{\beta}_A q' \implies \exists r'.\ (q',r') \in R \wedge r \xrightarrow{\beta^*}_B r'$*

A weak bisimulation over A and B is a relation R such that R and R^{-1} are simulations. We say that A and B are observationally equivalent and we write $A \sim B$ if for each state of A there is a weakly bisimilar state of B and conversely.

We use this definition to compare partial state and complete state semantics.

Theorem 1. $\gamma(B_1, \ldots, B_n) \sim \gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$

3 Partial State Semantics with Oracles

Let $\gamma(B_1, \ldots, B_n)$ be a system obtained as the composition of atomic components $B_i = (Q_i, P_i, \rightarrow_i)$ by using a set of interactions $\gamma \subseteq 2^P$ where $P = \bigcup_{i=1}^n P_i$. The corresponding partial state system $\gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$ consists of the components $B_i^{\perp} = (Q_i \cup Q_i^{\perp}, P_i \cup \{\beta_i\}, \rightsquigarrow_i)$ composed by using interactions in $\gamma^{\perp}$. As above, we take $\bigotimes_{i=1}^n (Q_i \cup Q_i^{\perp}) = Q^g \cup Q^p$. We also suppose that π is a priority for $\gamma(B_1, \ldots, B_n)$, and $\pi^{\perp}$ is its extension to partial states.

3.1 Basic Definitions and Properties

For a system $\gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$, a state $q \in Q^g \cup Q^p$ and an interaction $a \in \gamma$, we say that a is *enabled* at state q and we write $\texttt{enabled}(q, a)$, if the transition a is possible from state q. That is, $q \overset{a}{\rightsquigarrow} q'$ for some state q'. We say that a is *disabled* at state q and we write $\texttt{disabled}(q, a)$, if there is an atomic component in a *ready state* that prevents synchronization on a. That is, if $a = \{p_i\}_{i \in I}$ there is $i \in I, q_i \in Q_i$ such that $q_i \overset{p_i}{\not\rightsquigarrow}$.

For global states, $\texttt{disabled}(q, a)$ is equivalent to $q \overset{a}{\not\rightsquigarrow}$ and in particular we always have either $\texttt{disabled}(q, a)$ or $\texttt{enabled}(q, a)$. However, for partial states the status (disabled or enabled) of an interaction a at a given state may be unknown if some components involved in a are in busy states.

To compare partialness of states, we define a partial order relation over the states of composite components.

Definition 9 (State Ordering). *For $q, r \in Q^g \cup Q^p$, $q \leq r \iff \forall i \in \{1..n\}.\ (r_i = q_i \vee q_i \in Q_i^{\perp})$.*

For a given relation $\pi^{\perp}$, an oracle is a predicate $\mathcal{O}$ on $(Q^p \cup Q^g) \times \gamma$ used to strengthen the premises of the semantic rule for $\gamma^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$. Oracles are defined so that $\pi^{\perp}\gamma_{\mathcal{O}}^{\perp}(B_1^{\perp}, \ldots, B_n^{\perp})$ is observationally equivalent to $\pi\gamma(B_1, \ldots, B_n)$

where $\gamma^{\perp}_{\mathcal{O}}(B^{\perp}_1, \ldots, B^{\perp}_n)$ is the behavior restricted by the oracle. We introduce first a notion of composition with an oracle and in Subsection 3.2, we introduce oracles.

Definition 10 (Composite Components with Oracle). *Given an oracle $\mathcal{O}$ on $(Q^p \cup Q^g) \times \gamma$, we define $B \stackrel{def}{=} \gamma^{\perp}_{\mathcal{O}}(B^{\perp}_1, \ldots, B^{\perp}_n)$ as the transition system $(Q^p \cup Q^g, \gamma^{\perp}, \rightsquigarrow)$ where $\rightsquigarrow$ is the least set of transitions satisfying the rules*

$$\frac{a = \{p_i\}_{i \in I} \in \gamma \qquad \forall i \in I.\ q_i \stackrel{p_i}{\rightsquigarrow}_i q'_i \qquad \forall i \notin I.\ q_i = q'_i \qquad \mathcal{O}(q_1, \ldots, q_n, a)}{(q_1, \ldots, q_n) \stackrel{a}{\rightsquigarrow} (q'_1, \ldots, q'_n)}$$

$$\frac{q_i \stackrel{\beta_i}{\rightsquigarrow}_i q'_i}{(q_1, \ldots, q_i, \ldots, q_n) \stackrel{\beta_i}{\rightsquigarrow} (q_1, \ldots, q'_i, \ldots, q_n)}$$

The following proposition says that a system with an oracle $\mathcal{O}$ strongly simulates ([8]) a system with oracle $\mathcal{O}'$ such that $\mathcal{O} \implies \mathcal{O}'$.

Proposition 1. *Let $\mathcal{O}$ and $\mathcal{O}'$ be two oracles for the system $\gamma^{\perp}(B^{\perp}_1, \ldots, B^{\perp}_n)$, such that $\mathcal{O} \implies \mathcal{O}'$. They define two systems $B = \gamma^{\perp}_{\mathcal{O}}(B^{\perp}_1, \ldots, B^{\perp}_n) = (Q^g \cup Q^p, \gamma^{\perp}, \rightarrow_{\mathcal{O}})$ and $B' = \gamma^{\perp}_{\mathcal{O}'}(B^{\perp}_1, \ldots, B^{\perp}_n) = (Q^g \cup Q^p, \gamma^{\perp}, \rightarrow_{\mathcal{O}'})$. Every state of B is strongly similar to some state of B'.*

3.2 Oracles

We defines oracles parameterized by a dependency relation $\sqsubseteq$ on interactions. This relation contains $\pi^{\perp}$ but it need not be an order as shown below.

Definition 11 (Oracle). *A $\sqsubseteq$-oracle for a system $\gamma^{\perp}(B^{\perp}_1, \ldots, B^{\perp}_n)$ and a* dependency relation $\sqsubseteq \subseteq \gamma \times (Q^g \cup Q^p) \times \gamma$, *is a predicate $\mathcal{O}$ on $(Q^g \cup Q^p) \times \gamma$ such that:*

- *(Dependency Enforcement)*

$$\mathcal{O}(q, a) \implies \big(\forall a'.\ a \sqsubseteq_q a' \implies \texttt{disabled}(q, a') \vee \texttt{enabled}(q, a')\big)$$

- *(Soundness)* $q \in Q^g \implies \forall a \in \gamma.\ \mathcal{O}(q, a)$

The dependency enforcement condition means that the oracle allows execution of a from state q if the status (enabled or disabled) of the interactions a' that dominate a (*i.e.* $a \sqsubseteq_q a'$) is known.

Property 2. If $\sqsubseteq_1 \subseteq \sqsubseteq_2$ and if $\mathcal{O}$ is a $\sqsubseteq_2$-oracle, then it is a $\sqsubseteq_1$-oracle.

We will now define several $\pi^{\perp}$-oracles for the system $\gamma^{\perp}(B^{\perp}_1, \ldots, B^{\perp}_n)$ providing various degrees of parallelism and cost of implementation. There is a compromise to make between the degree of parallelism allowed by an oracle, and the cost for its implementation.

Ideal Oracle. The best possible oracle is defined by

$$\mathcal{O}_{ideal}(q, a) \iff \big(\forall a'.\ a \pi^{\perp}_q a' \implies \texttt{disabled}(q, a') \vee \texttt{enabled}(q, a')\big)$$

However, such an oracle is difficult to implement. It requires that at a given partial state q, the Engine is able to compute the relation $\pi_q^\perp$ which according to the definition of $\pi^\perp$ (Definition 7) boils down to computing the global state q' reachable from q. For this, in the general case, the Engine has to know the transition relation of the global state system.

Dynamic Oracle. We use now a dynamic approximation $\sqsubseteq^{dyn}$ of $\pi^\perp$. The reachability condition $q \overset{\beta^*}{\rightsquigarrow} q'$ in the definition of $\pi^\perp$ is replaced by a comparison $q \leq q'$, *i.e.* $a \sqsubseteq_q^{dyn} a' \iff \exists q' \in Q^g.\ q \leq q' \wedge a\pi_{q'}a'$. The dynamic oracle is defined by:

$$\mathcal{O}_{dyn}(q,a) \iff (\forall a'.\ a \sqsubseteq_q^{dyn} a' \implies \texttt{enabled}(q,a') \vee \texttt{disabled}(q,a'))$$

For the dynamic oracle, the Engine does not need a complete knowledge of the state of the system in order to compute $\sqsubseteq_q^{dyn}$ for a given partial state q.

Static Oracle. The static oracle $\mathcal{O}_{static}$ is defined via a static approximation $\sqsubseteq^{st}$ of $\pi^\perp$: $a \sqsubseteq_q^{st} a' \iff \exists q' \in Q^g.\ a\pi_{q'}a'$. We write $\sqsubseteq^{st}$ instead of $\sqsubseteq_q^{st}$ as the relation does not depend on q. The static oracle is defined by:

$$\mathcal{O}_{static}(q,a) \iff (\forall a'.\ a \sqsubseteq^{st} a' \implies \texttt{enabled}(q,a') \vee \texttt{disabled}(q,a'))$$

Lazy Oracle. The lazy oracle forbids all interactions from partial states. It waits for all the atomic components to finish their computation in order to know all the possible interactions. It is defined by $\mathcal{O}_{lazy}(q,a) \iff q \in Q^g$.

Proposition 2. *$\mathcal{O}_{ideal}$, $\mathcal{O}_{dyn}$, $\mathcal{O}_{static}$ and $\mathcal{O}_{lazy}$ are $\pi^\perp$-oracles and we have, $\mathcal{O}_{lazy} \implies \mathcal{O}_{static} \implies \mathcal{O}_{dyn} \implies \mathcal{O}_{ideal}$.*

The above result with Proposition 1 shows that these oracles provide an increasing degree of parallelism.

3.3 Correctness with Respect to Global State Semantics

The systems $\pi\gamma(B_1,\ldots,B_n)$ and $\pi^\perp\gamma_{\mathcal{O}}^\perp(B_1^\perp,\ldots,B_n^\perp)$ are observationally equivalent when $\mathcal{O}$ is a $\pi^\perp$-oracle.

Theorem 2. *Let π be a priority relation for the system $\gamma(B_1,\ldots,B_n)$ and $\mathcal{O}$ a $\pi^\perp$-oracle for the system $\gamma^\perp(B_1^\perp,\ldots,B_n^\perp)$. The systems $\pi\gamma(B_1,\ldots,B_n)$ and $\pi^\perp\gamma_{\mathcal{O}}^\perp(B_1^\perp,\ldots,B_n^\perp)$ are observationally equivalent.*

4 Distributed Semantics

4.1 Implementation

The model of BIP components with partial states is a first step towards a distributed implementation of BIP by separating internal computations from interactions. However, this model uses strong synchronization and therefore is still not directly implementable on arbitrary platforms where rendezvous is usually not available as a communication primitive.

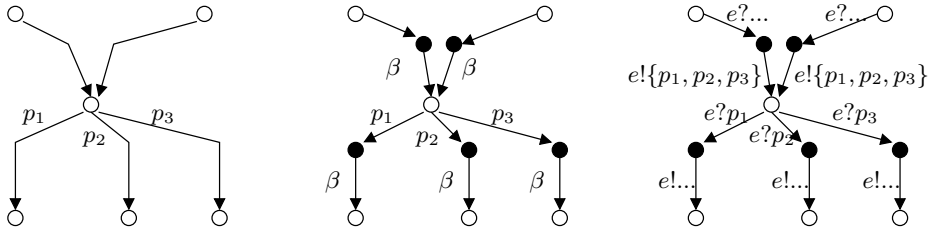


Fig. 2. Transformation from atomic BIP components (left) towards atomic components with partial states (middle) and io-machines (right)

We propose a second step towards a concrete distributed implementation of BIP components with partial states where multiparty interactions are replaced by asynchronous communication protocols (see Figure 2). The target model is *input-output systems* (io-systems) that are collections of parallel *input-output machines* (io-machines) communicating asynchronously by message passing through FIFO channels. This model is conceptually simple and directly encompasses primitives offered by languages used for modeling of distributed systems (such as SDL[7] or IO-automata[5]) or primitives usually available on distributed execution platforms (e.g. asynchronous execution of threads or processes, inter-process and inter-thread communication through FIFO queues, network protocols).

The principle of implementation is sketched in figure 3. Given $\pi^{\perp}\gamma^{\perp}(B_1^{\perp}, B_2^{\perp}, ..., B_n^{\perp})$ and a $\pi^{\perp}$-oracle $\mathcal{O}$, the implementation is an io-system consisting of io-machines B_i^{io} emulating the behavior of $B_i^{\perp}$ and an additional io-machine, the *Engine* $E(\gamma^{\perp}, \pi^{\perp}, \mathcal{O})$ realizing the coordination between them. Communication takes place only between the atomic components and the Engine, and never directly between different atomic components – this leads to an io-system with a centralized architecture.

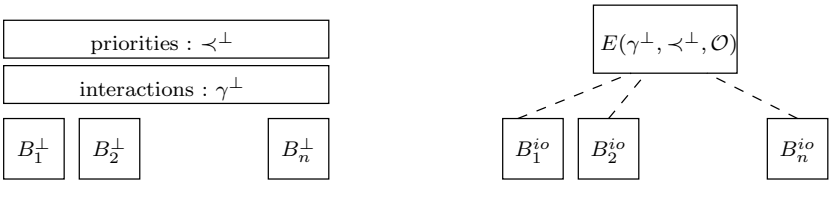


Fig. 3. Implementation: The Overall Structure

Formally, an io-system is a tuple $\mathcal{S} = (\mathcal{M}, Act, \{A_i = (Q_i, \hookrightarrow_i)\}_{i \in I})$ where

- $\mathcal{M}$ is a set of *messages*,
- *Act* is a set of actions α including *outputs* $j!m$ – output of the message $m \in \mathcal{M}$ to machine $j \in I$, *inputs* $j?m$ – input of message $m \in \mathcal{M}$ sent by machine $j \in I$ or *uninterpreted actions* a,

– $\{A_i = (Q_i, \hookrightarrow_i)\}_{i \in I}$ is a finite set of io-machines, where
  • Q_i is a finite set of states,
  • $\hookrightarrow_i \subseteq Q_i \times Act \times Q_i$ is a finite set of transitions labeled with actions.

States of io-systems are represented by configurations $\{(q_i, w_i)\}_{i \in I}$ where $q_i \in Q_i$ is a local state and $w_i \in (I \times \mathcal{M})^*$ is the FIFO-queue content of io-machine i. The semantics of io-systems is given as a labeled transition system on configurations. For each transition $q_i \overset{\alpha}{\hookrightarrow}_i q_i'$ of the io-machine i, we consider the following transitions on configurations corresponding respectively to input, output and uninterpreted actions:

– $\{..., (q_i, (j, m) \bullet w_i'), ...\} \overset{\tau}{\hookrightarrow} \{..., (q_i', w_i'), ...\}$ when $\alpha = j?m$,
– $\{..., (q_i, w_i), (q_j, w_j), ...\} \overset{\tau}{\hookrightarrow} \{..., (q_i', w_i), (q_j, w_j \bullet (i, m)), ...\}$ when $\alpha = j!m$
– $\{..., (q_i, w_i), ...\} \overset{a}{\hookrightarrow} \{..., (q_i', w_i), ...\}$ when $\alpha = a$,

The implementations of atomic components are io-machines obtained as follows. Whenever a ready state is reached, they output a message to the Engine containing (1) the sets of ports on which they are willing to interact and (2) their local ready state. Then, they wait for a notification from the Engine indicating the port selected for interaction. Depending on this port, they continue their execution. Formally, given $B_i^{\perp} = (Q_i \cup Q_i^{\perp}, P_i \cup \{\beta_i\}, \rightsquigarrow_i)$, its corresponding io-machine $B_i^{io} = (Q_i \cup Q_i^{\perp}, \hookrightarrow_i)$ has the same set of states as $B_i^{\perp}$ and transitions defined by the following rules (see Figure 2):

– $q_i \overset{e!(X,q_i')}{\hookrightarrow_i} q_i'$ *interaction request* whenever $q_i \overset{\beta_i}{\rightsquigarrow}_i q_i'$ and $X = \{p \mid q_i' \overset{p}{\rightsquigarrow}_i\}$
– $q_i \overset{e?p}{\hookrightarrow_i} q_i'$ *interaction notification* whenever $q_i \overset{p}{\rightsquigarrow}_i q_i'$

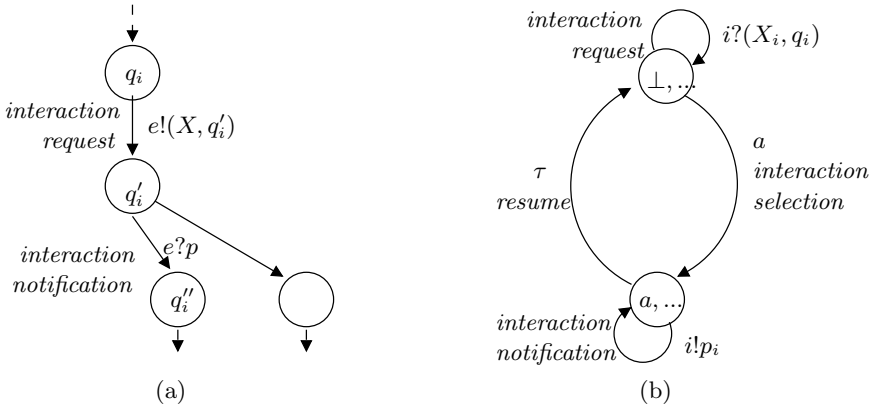


Fig. 4. Principle of Implementation: (a) io-machines for atomic components and (b) io-machine for the Engine

The Engine $E(\gamma^\perp, \pi^\perp, \mathcal{O})$ is an io-machine (see Figure 4) realizing the coordination between atomic io-machines for a given set of interactions $\gamma^\perp$, priorities $\pi^\perp$ and a $\pi^\perp$-oracle $\mathcal{O}$. Iteratively, the Engine receives and stores the sets of ports and the local states of components ready to interact. Depending on this information, it seeks a feasible interaction, which is maximal with respect to priorities and allowed by the oracle $\mathcal{O}$. If such an interaction exists, the Engine *executes* it by notifying sequentially, in some arbitrary order, all the involved components. Formally, given $\pi^\perp\gamma^\perp(B_1^\perp, B_2^\perp, ..., B_n^\perp)$ and an oracle $\mathcal{O}$, the Engine is the io-machine $(Q_e, \hookrightarrow_e)$ where

• $Q_e = (\gamma \cup \{\perp\}) \times \bigotimes_{i=1}^n 2^{P_i} \times \bigotimes_{i=1}^n (Q_i \cup \{\perp\})$ is the set of states of the form $(a^\perp, \mathbf{X}, \mathbf{q}^\perp)$ with $\mathbf{X} = (X_1, ..., X_n)$ and $\mathbf{q}^\perp = (q_1^\perp, ..., q_n^\perp)$ where

- $a^\perp \in \gamma \cup \{\perp\}$ is the interaction being currently executed, $\perp$ if none;
- $X_i \in 2^{P_i}$, is the set of ports on which component i is able to interact, empty if still busy;
- $q_i^\perp \in Q_i \cup \{\perp\}$ is the state q_i if component i is ready to interact, $\perp$ if still busy.

• $\hookrightarrow_e$ contains the following transitions

- $(\perp, \mathbf{X}, \mathbf{q}^\perp) \overset{i?X_i, q_i}{\hookrightarrow_e} (\perp, \mathbf{X}[X_i/i], \mathbf{q}^\perp[q_i/i])$ *interaction request*, stores information received from component i ready to interact.
- $(\perp, \mathbf{X}, \mathbf{q}^\perp) \overset{a}{\hookrightarrow_e} (a, \mathbf{X}, \mathbf{q}^\perp)$ *interaction selection*, whenever an interaction a exists such that $a \subseteq \cup_{i=1}^n X_i$, a is maximal with respect to priorities $\pi^\perp$ and a is allowed by the oracle $\mathcal{O}$ at state $\mathbf{q}^\perp$. It consists in executing the interaction and moving to a state from which all the components involved will be notified.
- $(a, \mathbf{X}, \mathbf{q}^\perp) \overset{i!p_i}{\hookrightarrow_e} (a, \mathbf{X}[\emptyset/i], \mathbf{q}^\perp[\perp/i])$ *interaction notification* and *cleanup* of the i component involved in the interaction a, that is when $a \cap X_i = \{p_i\} \neq \emptyset$,
- $(a, \mathbf{X}, \mathbf{q}^\perp) \overset{\tau}{\hookrightarrow_e} (\perp, \mathbf{X}, \mathbf{q}^\perp)$ *resume*, when all atomic components have been notified, that is $a \cap \cup_{i=1}^n X_i = \emptyset$. It consists in moving back to a state where requests are handled.

The correctness of the implementation is formally established by the following theorem.

Theorem 3. *Composite components with partial states* $\pi^\perp\gamma_{\mathcal{O}}^\perp(B_1, B_2, ..., B_n)$ *are weakly bisimilar to* $(\mathcal{M}, Act, \{B_1^{io}, ..., B_n^{io}, E(\gamma^\perp, \pi^\perp, \mathcal{O})\})$, i.e. *their io-system implementation where* τ *is a silent action.*

4.2 Experimental Results

Distributed Execution Platform. We have implemented the distributed semantics of BIP programs and included it into the BIP toolset[2]. This toolset is a collection of tools dedicated to execution and analysis of BIP programs currently providing:

• *A compilation chain that transforms BIP programs into C/C++ code.* Compilation relies on model-based technologies available for Java under the Eclipse platform. Starting from BIP programs, the compiler generates BIP models conforming to a full-fledged BIP meta-model developed using EMF[1]. On the models, we can apply source-to-source transformations as well as static analysis techniques. Finally, models are used to generate C/C++ code to be executed on a dedicated platform, as follows.

• *A platform for execution and analysis of the generated C/C++ code.* The execution platform includes an Engine and the associated software infrastructure for multithreaded execution of the C/C++ code. Each atomic component is assigned to a thread, the Engine being a thread itself. The Engine implements the distributed semantics and is parameterized by a dynamic or lazy oracle. Iteratively, the Engine computes feasible interactions available on ready components. Then, if such interactions exist and the oracle allows them, the Engine selects one for execution and notifies the involved components.

Benchmarks. We present two examples illustrating the application of the results on a prototype implementation. We evaluate for two different types of oracles, the degree of parallelism over time, measured as the number of simultaneously executing atomic components. Before providing experimental results, we analyze the relationship between degree of parallelism and parameters of the system.

To simplify the analysis, consider a system consisting of n atomic components always able to interact through their ports. We distinguish the following cases, illustrated in Figure 5:

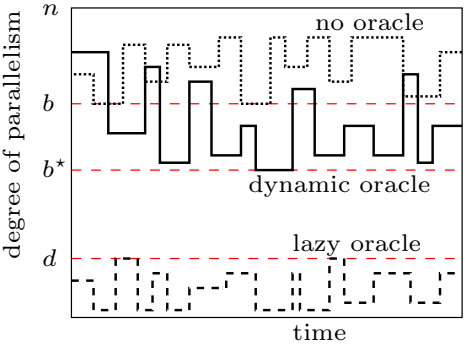


Fig. 5. Performance analysis

• For an implementation without oracle, the degree of parallelism is related to the minimal cardinality b of blocking subsets of atomic components. A subset of atomic components is *blocking* iff every interaction in the system requires at least one component of the subset to participate. Now, the degree of parallelism

[1] Eclipse Meta-modeling Framework.

l is such that $b \leq l \leq n$. In fact, whenever less than b components are running some interaction is possible and the Engine can eventually launch it;

• For an implementation with the lazy oracle, the maximal degree of parallelism is related to the maximal degree of interaction d, that is the maximal number d of components involved in a single interaction. In this case, the degree of parallelism l is such that $0 \leq l \leq d$. Interactions can be executed only from global states so there is no possibility of concurrency between interactions - the Engine is not able to keep running more than d atomic components at time;

• Finally, for dynamic oracles, the degree of parallelism is related again to the minimal cardinality $b^\star$ of some particular blocking sets of atomic components, the ones which block *all the maximal* interactions. We have $b^\star \leq b$ and the degree of parallelism l achieved in this case is such that $b^\star \leq l \leq n$. Using a similar reasoning as in the case without oracle, whenever less than $b^\star$ components are running, there should exist a maximal interaction ready and the Engine can eventually lauch it.

As a first benchmark, we consider a linear chain consisting of a set of identical components connected serially as shown in Figure 6. A component C_i has two ports, l_i and r_i. It has a single control state S_i, and two transitions labeled by l_i and r_i. The transition r_i is always enabled, its guard being *true*, whereas the transition l_i has a non-trivial guard g_i. We model broadcast from each component to its right neighbor by considering two types of interactions, 1) a set of singleton interactions consisting of the ports r_i; 2) a set of binary interactions $r_i l_{i+1}$ between the neighboring components, and 3) the priority $r_i \pi r_i l_{i+1}$ for the above interaction pair.

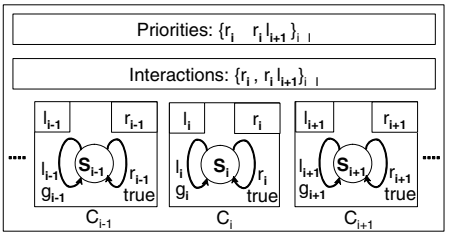


Fig. 6. The Linear Chain

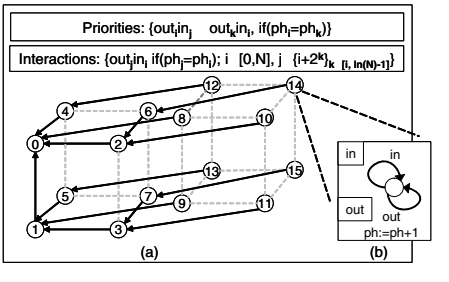


Fig. 7. The Parallel Adder

Our experiment considers a system with 25 such components. Each component executes 100 steps (transitions), getting busy for 50-60 milliseconds on an l transition and 5-6 milliseconds on an r transition respectively. We performed the experiment on a single-processor PC running linux. The busy times of the atomic components were simulated by *sleep* system calls. We measured the degree of parallelism in the system with respect to the execution time. Figure 8 shows the results obtained for dynamic and lazy oracles.

Without oracle, the degree of parallelism is 25 continuously. In fact, whenever a component is ready, it can continue alone on the r interaction and the Engine notifies it immediately. For the lazy oracle, the maximal degree of parallelism equals the maximal degree of an interaction, which is 2. Whenever an interaction takes place, the two participating atomic components are active simultaneously for the first 5-6 ms, after which only the atomic component performing the l transition remains busy for 50-60 ms. Therefore, the degree of parallelism stays at an average close to 1. Finally, for dynamic oracle, the minimal blocking set has cardinality 12 (as for a linear chain with n atoms, the minimal cardinality is $n/2$, when every alternate atoms are busy blocking all the maximal interactions). Hence, we have at least 12 atomic components executing at any time. The measured degree of parallelism in this case, remains in average higher than 15.

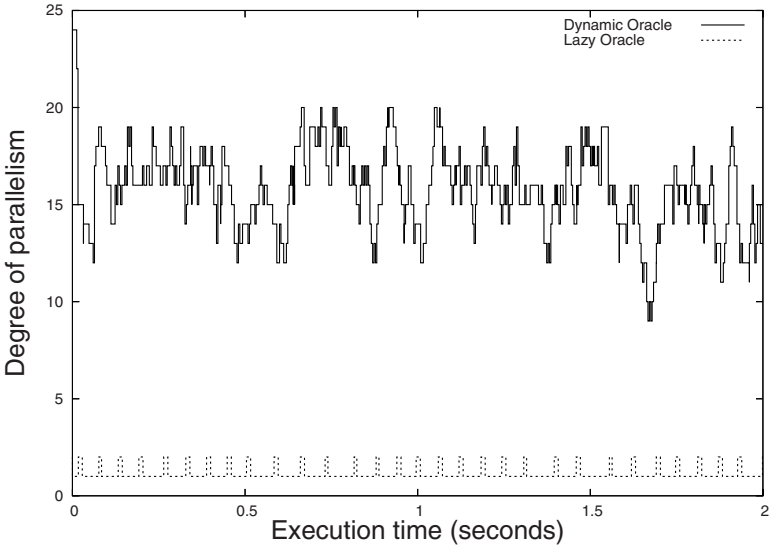


Fig. 8. Degree of Parallelism for Linear Chain

The second benchmark treats a parallel adder originally presented in [9], which adds 2^m values in a hypercube multi-processor machine. When the algorithm begins, the nodes hold the values to be added. On termination, the node labeled 0 contains their sum. Figure 7 presents the BIP model of a pipelined parallel-adder in a 4-dimensional hypercube with 2^4 nodes. Each node is modeled as a BIP component with ports *in* and *out*, labeling two transitions from a single control state, as shown in Figure 7(b). It also contains an array of values to be added (not shown on the figure) and the variable *ph* which records the index of current running addition on that node.

For each addition, every node receives partial addition results from its predecessors, adds them to its own value, sends the resulting sum to its unique successor and increments its *ph* variable. Communications between nodes are

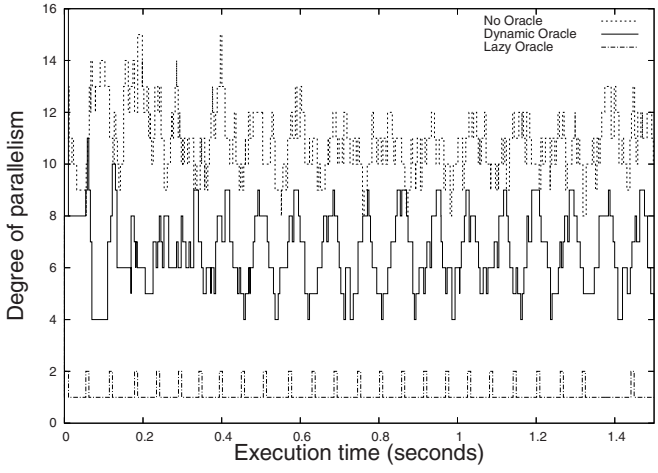


Fig. 9. Degree of Parallelism for Parallel Adder

modeled as interactions between the *out* port of a node and the *in* port of its successor, with a transfer of value from the node to the successor. Priorities are used to enforce correct order of the computation, *i.e.* a node cannot perform an *out* unless it has synchronized through its *in* port with all its predecessors. The final result of every addition is generated by the root node labeled 0.

The degrees of parallelism achieved, respectively without oracle and with lazy and dynamic oracles, are shown in Figure 9. Without oracle, the degree of parallelism is in average equal to 10. Let us notice that, without oracle, the functional behavior is completely wrong as priorities are used to enforce the right order of computation between nodes. With the lazy oracle, the maximal degree of parallelism equals the maximal degree of interaction which is 2. However, due to specific timing constraints on the execution of *in* and *out* transitions, the degree of parallelism stays in average close to 1. Finally, the dynamic oracle achieves a much better performance with an average degree of parallelism equal to 7.

5 Conclusion

We study a distributed implementation method for BIP, a framework for the description of component-based heterogeneous systems. BIP offers two powerful mechanisms for composing components by using interactions and priorities. The combination of interactions and priorities is expressive enough to express usual composition operators of other languages as shown in [3]. In particular to model broadcast, interactions do not suffice and other operators such as restrictions or priorities are needed. Furthermore, priorities are essential for describing scheduling policies, run-to-completion execution, urgency in real-time systems [6]. The proposed implementation method is quite general and can be easily adapted to other languages.

A key innovative idea is the translation of languages based on global state semantics to observationally equivalent distributed models from which implementation is straightforward. The decomposition of the translation in two steps allows separation of concerns in solving two main problems: the definition of partial state semantics and the expression of composition in terms of message passing primitives. Operational semantics provide an adequate framework for formalizing the translation. The models are obtained by successive refinements that preserve observational equivalence.

The main results show that whenever priorities are needed to express coordination between components, the operational semantics rules should be strengthened to take into account dependency between interactions. Oracles are very simple controllers enforcing preservation of semantics. Maximal parallelism is achieved for dynamic oracles allowing interaction as soon as possible. Nonetheless, these oracles may entail considerable computational overhead. As illustrated by experimental results the degree of parallelism depends on the type of the oracle and topology of the interactions.

There are many open problems to be investigated in the proposed framework for distributed implementation. These include the preservation of specific classes of properties, and less centralized implementations for the Engine.

References

1. Basu, A., Bidinger, P., Bozga, M., Sifakis, J.: Distributed semantics and implementation for systems with interaction and priority. Technical report, Verimag, Centre Équation, 38610 Gières (March 2008)
2. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: SEFM, pp. 3–12 (2006)
3. Bliudze, S., Sifakis, J.: The algebra of connectors – structuring interaction in BIP. In: EmSoft, pp. 11–20 (2007)
4. Chandy, K.M., Lamport, L.: Distributed snapshots: Determining global states of distributed systems. ACM Trans. Comput. Syst. 3(1), 63–75 (1985)
5. Garland, S.J., Lynch, N.A.: The ioa language and toolset: Support for designing, analyzing, and building distributed systems. Technical Report MIT/LCS/TR-762, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA (August 1998)
6. Gößler, G., Sifakis, J.: Priority systems. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 314–329. Springer, Heidelberg (2004)
7. ITU-T. Recommendation Z.100. Specification and Description Language (SDL). Technical Report Z-100, International Telecommunication Union – Standardization Sector, Genève (November 1999)
8. Milner, R.: Communication and concurrency. Prentice Hall International (UK) Ltd., Hertfordshire (1995)
9. Quinn, M.J.: Designing efficient algorithms for parallel computers. McGraw-Hill, Inc., New York (1986)