Optimized Distributed Implementations of Timed Component-based Systems

Ahlem Triki, Jacques Combaz, Saddek Bensalem

Verimag Research Report n^o TR-2015-7

August 10, 2015

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UJF
Centre Équation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Optimized Distributed Implementations of Timed Component-based Systems

AUTEURS = Ahlem Triki, Jacques Combaz, Saddek Bensalem

August 10, 2015

Abstract

Distributed implementation of real-time systems has always been a challenging task. The coordination of components executing on a distributed platform has to be ensured by complex communication protocols taking into account their timing constraints. We propose a novel method for distributed implementation of the application software formally expressed in Behavior, Interaction, Priority (BIP). A BIP model consists of a set of components, subject to timing constraints, and synchronizing through multiparty interactions. The proposed method transforms BIP models into Send/Receive BIP models that operate using asynchronous message passing. Send/Receive BIP models include additional components called schedulers that observe atomic components states. Based on these observations, the schedulers are required to plan as soon as possible the execution of interactions. We propose a method that optimizes the number of observed components, and thus reduces the number of exchanged messages.

Keywords: Component-based modeling, BIP, Distributed real-time systems, Source-to-source transformations.

Reviewers:

How to cite this report:

```
@techreport {TR-2015-7,
  title = {Optimized Distributed Implementations of Timed Component-based Systems},
  author = { Ahlem Triki, Jacques Combaz, Saddek Bensalem },
  institution = {{Verimag} Research Report},
  number = {TR-2015-7},
  year = {}
}
```

1 Introduction

Correct design and implementation of distributed systems has always been a challenging task. The complexity of such systems comes from multiple factors such as non-determinism, race conditions and asynchronous communication. Considering real-time systems is even more challenging in distributed context. Indeed, meeting timing constraints may fail while executing on a distributed platform since communications may take much more time than it is allowed by the timing constraints.

Model-based design is a promising approach that is based on a chain of steps starting from a model (specification) and ending up with an implementation on a given platform. In this paper, we focus on models defined using BIP framework [1]. BIP is a component-based framework for building real-time systems based on a rigorous formal semantics. A BIP component is essentially described by a timed automaton [2] whose transitions are labeled by ports. BIP encompasses multiparty interactions for synchronizing components and dynamic priorities for scheduling between interactions. An interaction is a synchronization (rendez-vous) between a subset of ports. Currently, BIP models are executed using centralized schedulers that implement the semantics of BIP by executing interactions either in a sequential [3] or parallel [4] way.

In this paper, we are interested in concurrent execution of BIP models, where interactions are scheduled concurrently. When two interactions share a component, we say that they are conflicting. We show that scheduling an interaction safely requires observing components participating in conflicting interactions. We provide a method for optimizing the number of observed components. In fact, we define for each interaction a predicate on global states that characterizes components that could not be observed by the interaction. The satisfiability of this predicate is checked using static analysis techniques. In particular, we are interested in the method presented in [5] for compositional verification of BIP models.

We present a method for distributed implementation of BIP models. Our method is based on source-to-source transformations of BIP models into Send/Receive BIP models in which components communicate through asynchronous message passing. Send/Receive BIP models consist on transforming components of BIP models into Send/Receive components that communicate with additional components called schedulers responsible for scheduling interactions. In case of conflicting interactions, additional component called reservation protocol is refereed to resolve such conflict. A second transformation concerns generating C++ executable code for each component of the Send/Receive models.

We conduct experiments on an application consisting on simulation of a set of collaborating robots implemented using our transformation method. We use our method for optimizing observed components and we show net improvement of the application's performance in terms of exchanged messages number.

The rest of the paper is structured as follows. In Section 2 we present the basic semantics model of BIP. In section 3, we provide safety condition for scheduling interactions concurrently. The method for optimizing observed components is described in section 4. We describe the transformation of BIP models into Send/Receive BIP models in Section 5. Experimental results are presented in Section 6. Related work is discussed in Section 7. Finally, we conclude in Section 8.

2 Basic Semantic Model of BIP

In this section, we present the operational *global state* semantics of BIP [1]. BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. In this paper we do not consider priorities.

2.1 Atomic Components

An *atomic component* is essentially a timed automaton [2] labeled by ports used for communication among different components.

DEFINITION 1 *An* atomic component B *is defined by the tuple* $B = (L, P, \mathsf{C}, T, \mathsf{tpc})$ *where* L *is a finite set of locations,* P *is a finite set of ports,* C *is a set of local clocks, and* $T \subseteq L \times (P \times G(\mathsf{C}) \times 2^{\mathsf{C}}) \times L$ *is a set of transitions labeled with a port, a timing constraint and a subset of clocks to be reset.* $\mathsf{tpc} : L \longrightarrow G(\mathsf{C})$ *assigns to each location* $\ell \in L$ *a time progress condition* $\mathsf{tpc}_\ell \in G(\mathsf{C})$. $G(\mathsf{C})$ *is the set of timing constraints*

that are defined according to the following grammar: $\mathsf{tc} := \texttt{true} \mid \texttt{false} \mid c \sim k \mid \mathsf{tc} \wedge \mathsf{tc}$*, with* $c \in \mathsf{C}$*,* $k \in \mathbb{Z}_{\geq 0}$ *and* $\sim \in \{\leq, =, \geq\}$*. Time progress conditions are timing constraints where* $\sim$ *is restricted to* $\{\leq\}$*.*

Notice that any timing constraint tc can be put into a conjunction of the form:

$$\mathsf{tc} = \bigwedge_{c \in \mathsf{C}} l_c \leq c \leq u_c, \tag{1}$$

such that for all $c \in \mathsf{C}$, $l_c \in \mathbb{Z}_{\geq 0}$ and $u_c \in \mathbb{Z}_{\geq 0} \cup \{+\infty\}$. We denote by $L[\mathsf{tc}](c) = l_c$ and $U[\mathsf{tc}](c) = u_c$ the lower bound and the upper bound of timing constraint tc over clock c respectively.

In practice, an atomic component can be extended with variables which are used to store (private) local data. Variables can be exported through ports allowing exchange of data among components. Moreover, each component transition can be associated with a boolean condition specifying for which values of the local variables it is enabled, and an (internal) update function triggered along with transition execution which modifies values of variables.

Before recalling the semantics of an atomic component, we first fix some notations. Given a set of clocks C, a *valuation* $t : \mathsf{C} \to \mathbb{R}_{\geq 0}$ is a function associating with each clock c its value $t(c) \in \mathbb{R}_{\geq 0}$. We denote by $\mathcal{T}(\mathsf{C})$ the set of valuations of clocks in C. Given a subset of clocks $\mathsf{C}' \subseteq \mathsf{C}$ and a clock value $l \in \mathbb{R}_{\geq 0}$, we denote by $t[\mathsf{C}' \mapsto l]$ the valuation that coincides with t for all clocks $c \in \mathsf{C} \setminus \mathsf{C}'$, and that associates l to all clocks $c \in \mathsf{C}'$. The notation $t + \delta$ represents a new valuation t' defined by $(t + \delta)(c) = t(c) + \delta$ for any $c \in \mathsf{C}$.

DEFINITION 2 *The semantics of an atomic component* $B = (L, P, \mathsf{C}, T, \mathsf{tpc})$ *is defined as the labeled transition system* $(Q_B, P \cup \mathbb{R}_{\geq 0}, \to)$*, where* $Q_B = L \times \mathcal{T}(\mathsf{C})$ *is the set of states,* $\to \subseteq Q_B \times (P \cup \mathbb{R}_{\geq 0}) \times Q_B$ *is the set of labeled transitions satisfying the following rules:*

- $(\ell, t) \xrightarrow{p} (\ell', t[r \mapsto 0])$ *if* $\exists\, \tau = (\ell, p, \mathsf{tc}, r, \ell') \in T \wedge \mathsf{tc}(t)$ *(jump step).*
- $(\ell, t) \xrightarrow{\delta} (\ell, t + \delta)$ *if* $\mathsf{tpc}_\ell(t + \delta)$*(delay step).*

An atomic component B can execute a transition $\tau = (\ell, p, \mathsf{tc}, r, \ell')$ from a state (ℓ, t) if its timing constraint is met by the valuation t. The execution of τ corresponds to moving from control location ℓ to ℓ', and reseting clocks of r. From state (ℓ, t), B can also wait for $\delta > 0$ time units if the time progress condition tpc_ℓ stays $\texttt{true}$. Waiting for δ time units increases all the clock values by δ. Notice that the execution of transitions is instantaneous and time elapses only on states.

The semantics of timed automata presented here is slightly different from the one found in [2], as we consider time progress conditions instead of invariants. In contrast to invariant, an atomic component B may reach a state (ℓ, t) violating the corresponding time progress condition tpc_ℓ. In this case B cannot wait and is forced to execute a transition from (ℓ, t). In the following we consider systems that cannot reach states violating time progress conditions. A state (ℓ, t) from which B can neither execute a transition nor wait is a *timelock* [6]. In the following, we also consider systems that cannot reach timelocks.

EXAMPLE 1 *Figure 1 shows an atomic component* B *corresponding to a task that is processing items cyclically. It takes a new item to process via port* **take***, and give it back after processing via port* **give***. Transition* $\tau_1 = (\ell_1, \mathbf{take}, c = P, \{c\}, \ell_2)$ *executes when clock* c *reaches* P *and, resets* c *so as to measure the time it takes to give back an item after taking it. We assume that* B *takes exactly* E *time unit to process an item once it is taken. We also assume that once an item is processed, it can be kept by* B *for at most* K *time units before giving it back. For instance,* B *is a machine that is processing items at room temperature and they need to be kept cold or hot. This is represented by timing constraint* $E \leq c \leq D$ *for transition* $\tau_2 = (\ell_2, \mathbf{give}, E \leq c \leq D, \{c\}, \ell_1)$*, where* $D = E + K$*. To enforce the execution of* τ_2 *before* c *reaches* D*, we also consider the time progress condition* $\mathsf{tpc}_{\ell_2} = c \leq D$ *for* ℓ_2*.*

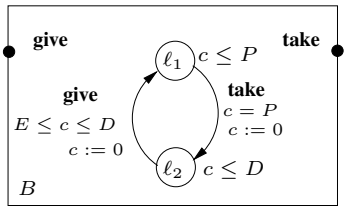

Figure 1: An atomic component.

DEFINITION 3 *Let $B = (L, P, \mathsf{C}, T, \mathsf{tpc})$ an atomic component with semantics $(Q_B, P \cup \mathbb{R}_{\geq 0}, \rightarrow)$. We say that B has non-decreasing deadlines if for each state (ℓ, t) in Q_B, for each transition $\tau = (\ell, p, \mathsf{tc}, r, \ell')$ in T such that $\mathsf{tc}(t)$ is* `true` *and for each $\delta \in \mathbb{R}_{\geq 0}$ we have:*

$$(\ell, t) \xrightarrow{\delta} (\ell, t+\delta) \Rightarrow (\ell, t) \xrightarrow{p} (\ell', t') \xrightarrow{\delta} (\ell', t'+\delta)$$

The following proposition gives syntactic conditions on time progress conditions and timing constraints that ensure satisfaction of the non-decreasing deadlines property.

PROPOSITION 1 *An atomic component $B = (L, P, \mathsf{C}, T, \mathsf{tpc})$ such that all its transitions $\tau = (\ell, p, \mathsf{tc}, r, \ell') \in T$ satisfy the following syntactic conditions $\begin{cases} U[\mathsf{tpc}_\ell](c) \leq U[\mathsf{tpc}_{\ell'}](c) & \text{if } c \notin r \\ U[\mathsf{tpc}_\ell](c) \leq U[\mathsf{tpc}_{\ell'}](c) + L[\mathsf{tc}](c) & \text{if } c \in r \end{cases}$ has non-decreasing deadlines.*

Proof 1 *We have $\tau = (\ell, p, \mathsf{tc}, r, \ell')$ and $(\ell, t) \xrightarrow{p} (\ell', t')$, we need to prove that $\forall\ c \in \mathsf{C}$ we have $t'(c) + \delta \leq U[\mathsf{tpc}_{\ell'}](c)$. That is, δ is enabled from state (ℓ', t').*

First, as $\mathsf{tc}(t)$ is true then we have:

$$\forall c \in \mathsf{C},\ t(c) \geq L[\mathsf{tc}](c) \tag{2}$$

Moreover, as $(\ell, t) \xrightarrow{\delta} (\ell, t+\delta)$ then we have:

$$\forall c \in \mathsf{C},\ L[\mathsf{tc}](c) \leq t(c) + \delta \leq U[\mathsf{tpc}_\ell](c) \tag{3}$$

- *if $c \notin r$ we have $t'(c) = t(c)$. According to the syntactic conditions of Proposition 1 and inequation (3), we have $t'(c) \leq U[\mathsf{tpc}_{\ell'}](c)$*
- *if $c \in r$ we have $t'(c) = 0$. According to the syntactic conditions of Proposition 1 and inequations (2) and (3), we have $\delta \leq U[\mathsf{tpc}_{\ell'}](c)$.*

.

In this paper we consider atomic components that satisfy the non-decreasing deadlines property.

2.2 BIP models

A BIP model is built from a set of n atomic components $\{B_i = (L_i, P_i, \mathsf{C}_i, T_i, \mathsf{tpc}_i)\}_{i=1}^n$, such that their respective sets of ports and clocks are pairwise disjoint; i.e., for any two $i \neq j$ from $\{1, \ldots, n\}$, we have $P_i \cap P_j = \emptyset$ and $\mathsf{C}_i \cap \mathsf{C}_j = \emptyset$.

DEFINITION 4 *An* interaction *between atomic components $\{B_i\}_{i=1}^n$ is a subset of ports $a \subseteq P$, such that it contains at most one port of every component, that is, $|a \cap P_i| \leq 1$ for all $i \in \{1, \ldots, n\}$.*

Since an interaction a uses at most one port of every component, we simply denote $a = \{p_i\}_{i \in I}$, where $I \subseteq \{1, \ldots, n\}$ and $p_i \in P_i$ for all $i \in I$. A component B_i is participating in a if $i \in I$. We denote by $part(a) = \{B_i \mid i \in I\}$ the set of components participating in a.

DEFINITION 5 *We denote by* $B \stackrel{def}{=} \gamma(B_1, \ldots, B_n)$ *the BIP model obtained by applying a set of interactions* γ *to the set of atomic components* $\{B_i = (L_i, P_i, \mathsf{C}_i, T_i, \mathsf{tpc}_i)\}_{i=1}^n$. *It is defined by the atomic component* $B = (L, \gamma, \mathsf{C}, T_\gamma, \mathsf{tpc})$, *where* $L = L_1 \times \ldots \times L_n$, $\mathsf{C} = \bigcup_{i=1}^n \mathsf{C}_i$, $\mathsf{tpc}(\ell) = \bigwedge_{i \in n} \mathsf{tpc}_{\ell_i}$, *A transition* $\tau = (\ell, a, \mathsf{tc}, r, \ell')$ *from* $\ell = (\ell_1, \ldots, \ell_n)$ *to* $\ell' = (\ell'_1, \ldots, \ell'_n)$ *is in* T_γ *iff (1)* $a = \{p_i\}_{i \in I} \in \gamma$, *(2) for all* $i \notin I$ $\ell'_i = \ell_i$ *and (3) and there exist transitions* $\tau_i = (\ell_i, p_i, \mathsf{tc}_i, r_i, \ell'_i)$ *of* B_i, $i \in I$, *such that* $\mathsf{tc} = \bigwedge_{i \in I} \mathsf{tc}_i$, $r = \bigcup_{i \in I} r_i$.

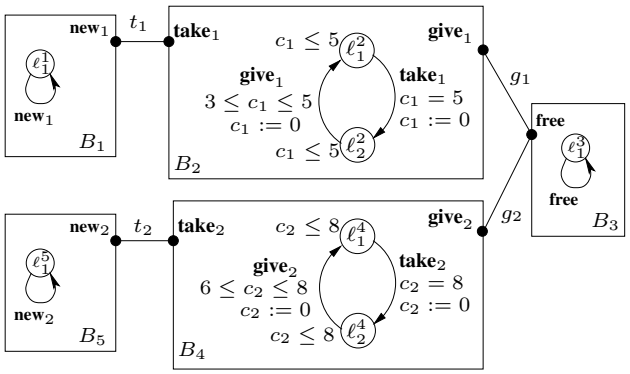

Figure 2: Example of a BIP model.

EXAMPLE 2 *Figure 2 illustrates a BIP model* $\gamma(B_1, B_2, B_3, B_4, B_5)$. *Components* B_2 *and* B_4 *are instances of component* B *of Figure 1 with E=3, D=P=5 for* B_2 *and E=6, D=P=8 for* B_4. *The set of interactions* γ *is* $\{t_1, t_2, g_1, g_2\}$, *where* $t_1 = \{\mathbf{new}_1, \mathbf{take}_1\}$, $t_2 = \{\mathbf{new}_2, \mathbf{take}_2\}$, $g_1 = \{\mathbf{give}_1, \mathbf{free}\}$ *and* $g_2 = \{\mathbf{give}_2, \mathbf{free}\}$. *Note that all components* $B_1, \ldots, B_5$ *have non-decreasing deadlines since they satisfy the syntactic conditions given in Proposition 1.*

Initially, the system is at state $(\ell, 0)$ *where* $\ell = (\ell_1^1, \ell_1^2, \ell_1^3, \ell_1^4, \ell_1^5)$. *At this state, the time progress condition and the timing constraint in* B_1 *impose that* t_1 *executes after a delay of 5 time units. Then due to the time progress condition and the timing constraint in* B_2, t_2 *executes after a delay of 3 time units. Then,* g_1 *executes after a delay* $\delta_1 \in [0, 2]$, *then* g_2 *after a delay* δ_2 *such that* $\delta_1 + \delta_2 \in [6, 8]$.

3 Safety Condition for Scheduling Interactions

Since we target distributed settings, we assume concurrent execution of interactions. However, if two interactions are simultaneously enabled, they cannot always run in parallel without breaking semantics of the global state model. Consider two interactions a and b that involve non disjoint sets of components (i.e. $part(a) \cap part(b) \neq \emptyset$). Clearly, a and b cannot execute in parallel because they share at least one component. Such a situation is called a *conflict*.

DEFINITION 6 *Let* $\gamma(B_1, \ldots, B_n)$ *be a BIP model. We say that two interactions* a *and* b *of* γ *are in* conflict *denoted by* $a \# b$, *iff there exists an atomic component* $B_i \in part(a) \cap part(b)$ *that has two transitions* $\tau_1 = (\ell, p_1, \mathsf{tc}_1, r_1, \ell'_1)$ *and* $\tau_2 = (\ell, p_2, \mathsf{tc}_2, r_2, \ell'_2)$ *from the same control location* ℓ *such that* $p_1 \in a$ *and* $p_2 \in b$.

Note that conflicts as defined in Definition 6 are an over approximation of conflicts since some conflicts may not be reachable due to system dynamics. A special case of conflict is when two interactions a and b share a common port, that is, $a \cap b \neq \emptyset$. Consider again the example from Figure 2. The interactions t_1 and t_2 are not conflicting with g_1 and g_2. However, g_1 and g_2 are conflicting because they share port **free** of component B_3.

Let a be an interaction. When a is scheduled, scheduling any interaction b that is conflicting with a (i.e., $b \# a$) needs to be blocked until a executes. If a is scheduled to execute in δ_a time units, any component

$B_i \in part(a)$ can only participate to a and is forced to wait for δ_a time units, which needs to be allowed by its corresponding time progress condition. Moreover, any component $B_j \in part(b)$ may participate to any interaction other than b. However, when the only enabled interaction is b, B_j is forced to wait for δ_a. In this case and if the time progress condition of B_j does not allow to wait for δ_a time units, scheduling a in δ_a introduces a timelock in B_j. That is, B_j cannot execute during δ_a time units but at the same time cannot wait for δ_a time units.

Problem Formulation

We denote by $\mathcal{L}_{p_i}$ the set of locations of component B_i enabling port p_i, defined as follows: $\mathcal{L}_{p_i} = \{\ell_i \mid \wedge \tau_i = (\ell_i, p_i, \ell_i') \in T_i\}$. Let $a = \{p_i\}_{i \in I}$ be an interaction. We denote by $\mathcal{L}_a$ the set of locations configurations enabling interaction a defined as follows: $\mathcal{L}_a = \bigotimes_{p_i \in a} \mathcal{L}_{p_i}$. We define the predicate $\mathsf{sched\text{-}tc}_a[\ell_a]$ characterizing clocks valuations for which a could be scheduled from the configuration $\ell_a \in \mathcal{L}_a$. It is defined as follows:

$$\mathsf{sched\text{-}tc}_a[\ell_a] = \bigwedge_{\{\ell_i \in \ell_a \mid \tau_i = (\ell_i, p_i, \mathsf{tc}_i, r_i, \ell_i') \in T_i \wedge p_i \in a\}} \mathsf{tc}_{\tau_i} \wedge \mathsf{tpc}_{\ell_i}.$$

In order to safely schedule a from the configuration ℓ_a in δ_a time units, we need to verify that each component B_j participating in a conflicting interaction b is allowed to wait for δ_a. As we consider components that satisfy the non-decreasing deadline property, we need only to check the time progress condition of the current state of B_j: if B_j can wait for δ_a from the current state, it will be able to do so even if it changes its state by executing other interactions.

We call *observed* components of an interaction a, denoted by $obs(a)$, the components that are not participating in a but that need to be observed in order to safely schedule a. It is defined as follows:

$$obs(a) = \bigcup_{a \# b} part(b) \setminus part(a).$$

We denote by $\mathsf{safe\text{-}tc}_a[\ell_a]$ the predicate characterizing the valuations of clocks for which a could be safely scheduled from the configuration ℓ_a. It is defined as follows:

$$\mathsf{safe\text{-}tc}_a[\ell_a] = \mathsf{sched\text{-}tc}_a[\ell_a] \bigwedge_{B_j \in obs(a)} \mathsf{tpc}_{\ell_j}.$$

In order to safely schedule interaction a in δ_a from a state (ℓ, t) where the location ℓ enables a from configuration ℓ_a, we have to check that $\mathsf{safe\text{-}tc}_a[\ell_a](t + \delta_a)$ evaluates to true.

4 Observed Components Number Reduction

In this section, we propose to use static analysis techniques in order to reduce, for each interaction a, the number of observed components. Intuitively, a component B_j could be reduced from the set $obs(a)$ if for each configuration ℓ_a from which a is enabled, each valuation of clocks that satisfies $\mathsf{sched\text{-}tc}_a[\ell_a]$ satisfies also the time progress condition of locations enabling a port p_j of B_j participating in each interaction b that is conflicting with a, i.e. $b \# a$. This could be expressed in terms of predicate on states enabling both a and each port $p_j \in b$, whose satisfiability allows reducing B_j from the set $obs(a)$. To this end, we define the predicate $\mathsf{reduce}_a(B_j)$ indicating whether the component B_j could not be observed by a, and thus be reduced from $obs(a)$. We denote by $\mathsf{confinter}_a(B_j)$ the set of interactions that are conflicting with a and involving component B_j, defined as follows: $\mathsf{confinter}_a(B_j) = \{b \in \gamma \mid b \# a \wedge B_j \in part(b)\}$. In the following, we define the predicate $\mathsf{reduce}_a(B_j)$.

DEFINITION 7 *Let $\gamma(B_1, \ldots, B_n)$ be a BIP model. We denote by C be the set of clocks in the BIP model. Consider an interaction a in γ and an observed component B_j in $obs(a)$. We define $\mathsf{reduce}_a(B_j)$ the predicate indicating whether B_j could be reduced from $obs(a)$ as follows:*

$$\mathsf{reduce}_a(B_j)$$

$$\equiv$$
$$\forall\, \ell_a \in \mathcal{L}_a, \forall \ell_j \in \mathcal{L}_{p_j} \mid p_j \in b \wedge b \in \mathsf{confinter}_a(B_j)$$
$$\forall t \in \mathcal{T}(\mathsf{C}),\ \forall \delta > 0$$
$$\mathsf{sched\text{-}tc}_a[\ell_a](t+\delta) \Rightarrow \mathsf{tpc}_{\ell_j}(t+\delta)$$

According to Definition 7, interaction a reduces the observation of component B_j, if for each configuration of locations $\ell_a \in \mathcal{L}_a$ enabling interaction a, for each location ℓ_j of component B_j enabling port p_j involved in interaction b such that $b\#a$, for each clocks valuation $t \in \mathcal{T}(\mathsf{C})$ and for each time step $\delta > 0$, we have $\mathsf{sched\text{-}tc}_a(\ell_a)(t+\delta) \Rightarrow \mathsf{tpc}_{\ell_j}(t+\delta)$. The latter implication specifies that there is no clocks valuation $t \in \mathcal{T}(\mathsf{C})$ that satisfies $\mathsf{sched\text{-}tc}_a[\ell_a]$ and not tpc_{ℓ_j}.

The predicate $\mathsf{reduce}_a(B_j)$ involves a non-constant variable δ and the clocks valuation t. Thus, static analysis techniques cannot be used at this step.

In order to obtain a static expression of $\mathsf{reduce}_a(B_j)$, we use explicit expressions of $\mathsf{sched\text{-}tc}_a[\ell_a]$ and tpc_{ℓ_j} to rewrite $\mathsf{reduce}_a(B_j)$.

Using (1), $\mathsf{sched\text{-}tc}_a[\ell_a]$ can be written into the following form: $\mathsf{sched\text{-}tc}_a[\ell_a] = \bigwedge_{c_a \in \mathsf{C}_a} l_{c_a}^{\ell_a} \leq c_a \leq u_{c_a}^{\ell_a}$, where C_a is the set of all clocks involved in $\mathsf{sched\text{-}tc}_a[\ell_a]$, and $l_{c_a}^{\ell_a}$ (resp. $u_{c_a}^{\ell_a}$) is the lower (resp. upper) bound value involving clock c_a in $\mathsf{sched\text{-}tc}_a[\ell_a]$. Similarly, tpc_{ℓ_j} can be written in the following form: $\mathsf{tpc}_{\ell_j} = \bigwedge_{c_j \in \mathsf{C}_j} c_j \leq d_{c_j}^{\ell_j}$, where C_j is the set of clocks of component B_j.

PROPOSITION 2 *Given an interaction a and a component $B_j \in obs(a)$, the predicate $\mathsf{reduce}_a(B_j)$ can be rewritten in the following form:*

$$\mathsf{reduce}_a(B_j)$$
$$\equiv$$
$$\forall\, \ell_a \in \mathcal{L}_a, \forall \ell_j \in \mathcal{L}_{p_j} \mid p_j \in b \wedge b \in \mathsf{confinter}_a(B_j)$$
$$\bigwedge_{c_j \in \mathsf{C}_j} \bigvee_{c_a \in \mathsf{C}_a} c_j - c_a \leq d_{c_j}^{\ell_j} - u_{c_a}^{\ell_a} \bigvee_{c_a \in \mathsf{C}_a} \bigvee_{c'_a \in \mathsf{C}_a} c_a - c'_a < l_{c_a}^{\ell_a} - u_{c'_a}^{\ell_a}$$

Proof 2 *We have to prove that:*

$$\forall t \in \mathcal{T}(\mathsf{C}), \forall \delta > 0$$
$$\mathsf{sched\text{-}tc}_a(\ell_a)(t+\delta) \Rightarrow \mathsf{tpc}_{\ell_j}(t+\delta)$$
$$\Leftrightarrow$$
$$\bigwedge_{c_j \in \mathsf{C}_j} \bigvee_{c_a \in C_a} c_j - c_a \leq d_{c_j}^{\ell_j} - u_{c_a}^{\ell_a} \bigvee_{c_a \in \mathsf{C}_a} \bigvee_{c'_a \in \mathsf{C}_a} c_a - c'_a < l_{c_a}^{\ell_a} - u_{c_a}$$

$\mathsf{sched\text{-}tc}_a[\ell_a](t+\delta) \Rightarrow \mathsf{tpc}_{\ell_j}(t+\delta)$

$\Leftrightarrow$ $\mathsf{tpc}_{\ell_j}(t+\delta) \vee \neg \mathsf{sched\text{-}tc}_a(\ell_a)(t+\delta)$

$\Leftrightarrow$ $\Big[\bigwedge_{c_j \in \mathsf{C}_j} t(c_j) \leq d_{c_j}^{\ell_j} - \delta\Big] \vee \Big[\bigvee_{c_a \in \mathsf{C}_a} t(c_a) > l_{c_a}^{\ell_a} - \delta\Big] \vee \Big[\bigvee_{c_a \in \mathsf{C}_a} t(c_a) > u_{c_a}^{\ell_a} - \delta\Big]$

$\Leftrightarrow$ $\Big[\bigwedge_{c_j \in \mathsf{C}_j} t(c_j) \leq d_{c_j}^{\ell_j} - \delta \bigvee_{c_a \in \mathsf{C}_a} t(c_a) > u_{c_a}^{\ell_a} - \delta\Big] \vee \Big[\bigvee_{c_a \in \mathsf{C}_a} t(c_a) < l_{c_a}^{\ell_a} - \delta \bigvee_{c_a \in \mathsf{C}_a} t(c_a) > u_{c_a}^{\ell_a} - \delta\Big]$

$\Leftrightarrow$ $\Big[\bigwedge_{c_j \in \mathsf{C}_j} \bigvee_{c_a \in \mathsf{C}_a} t(c_j) - t(c_a) \leq d_{c_j}^{\ell_j} - u_{c_a}^{\ell_a} - \delta\Big] \vee \Big[\bigvee_{c_a \in \mathsf{C}_a} \bigvee_{c'_a \in \mathsf{C}_a} t(c_a) < l_{c_a}^{\ell_a} - \delta \vee t(c'_a) > u_{c'_a} - \delta\Big]$

$\Leftrightarrow$ $\Big[\bigwedge_{c_j \in \mathsf{C}_j} \bigvee_{c_a \in \mathsf{C}_a} t(c_j) - t(c_a) \leq d_{c_j}^{\ell_j} - u_{c_a}^{\ell_a} - \delta\Big] \vee \Big[\bigvee_{c_a \in \mathsf{C}_a} \bigvee_{c'_a \in \mathsf{C}_a} \neg[t(c_a) \geq l_{c_a}^{\ell_a} - \delta \wedge t(c'_a) \leq u_{c'_a} - \delta]\Big]$

$\Leftrightarrow$ $\Big[\bigwedge_{c_j \in \mathsf{C}_j} \bigvee_{c_a \in \mathsf{C}_a} t(c_j) - t(c_a) \leq d_{c_j}^{\ell_j} - u_{c_a}^{\ell_a} - \delta\Big] \vee \Big[\bigvee_{c_a \in \mathsf{C}_a} \bigvee_{c'_a \in \mathsf{C}_a} \neg[t(c_a) - t(c'_a) \geq l_{c_a}^{\ell_a} - u_{c'_a}^{ell_a}]\Big]$

$\Leftrightarrow$ $\Big[\bigwedge_{c_j \in \mathsf{C}_j} \bigvee_{c_a \in \mathsf{C}_a} c_j - tc_a \leq d_{c_j}^{\ell_j} - u_{c_a}^{\ell_a} - \delta\Big] \vee \Big[\bigvee_{c_a \in \mathsf{C}_a} \bigvee_{c'_a \in \mathsf{C}_a} \neg[c_a - c'_a \geq l_{c_a}^{\ell_a} - u_{c'_a}]\Big]$.

We use static analysis techniques to check the satisfiability of $\mathsf{reduce}_a(B_i)$ as defined in Proposition 2, for each interaction a and for each for each component $B_j \in obs(a)$. In particular, we focus on the method presented in [5], for compositional verification of BIP models. This method relies on invariants computation. These invariants characterize an over-approximation of reachable states of BIP models. We do not detail here how to compute these invariants, interested readers may refer to [5] for more details.

Let J be the global invariant characterizing an over-approximation of reachable states of a BIP model. Verifying that $\mathsf{reduce}_a(B_j)$ holds is done by checking that $J \wedge \neg\mathsf{reduce}_a(B_j)$ is not satisfiable. It means that the intersection between the states violating the property $\mathsf{reduce}_a(B_j)$ and the states satisfying the invariant is empty. Thus, $\mathsf{reduce}_a(B_j)$ holds for all reachables states, since they all satisfy the global invariant J.

5 From BIP Models to Send/Receive BIP models

In this section, we explain our method for automated transformation of BIP models with multiparty interactions into *Send/Receive* BIP models involving only binary (Send/Receive) interactions that can be directly mapped on a distributed platform.

5.1 Send/Receive BIP model Architecture

In a Send/Receive BIP model, interactions are implemented by a protocol between the atomic components and a set of new components acting as schedulers, each one being responsible for a subset of interactions.

Intuitively, a Send/Receive BIP model is a set of independent components communicating through asynchronous message passing. It is formally defined as follows.

DEFINITION 8 *We say that $B^{SR} = \gamma^{SR}(B_1^{SR}, ..., B_n^{SR})$ is a send/Receive BIP Model iff we can partition the set of ports in B^{SR} into three sets P_s, P_r and P_u that are respectively the set of $send - ports$, $receive - ports$ and $unary - ports$, such that:*

- *Each interaction $a \in \gamma^{SR}$, is either (1) a Send/Receive interaction with $a = (s, r_1, r_2, \ldots, r_k)$, $s \in P_s$,*
  $r_1, \ldots, r_k \in P_r$ or, (2) a unary interaction $a = \{p\}$ with $p \in P_u$.
- *If s is a port in P_s, then there exists one and only one Send/Receive interaction $a \in \gamma^{SR}$ with $a = (s, r_1, r_2, \ldots, r_k)$ and all ports $r_1, \ldots, r_k$ are receive-ports. We say that $r_1, r_2, \ldots, r_k$ are the receive-ports associated to s.*
- *If $a = (s, r_1, \ldots, r_k)$ is a Send/Receive interaction in γ^{SR} and s is enabled at some global state of B^{SR}, then all its associated receive-ports $r_1, \ldots, r_k$ are also enabled at that state.*

In a Send/Receive BIP model, messages are sent through send-ports to receive-ports. A send-port has an associated set of receive-ports. Moreover, receive-ports must be ready to receive any message sent by the corresponding send-port.

Let $B = \gamma(B_1, \ldots, B_n)$ be an input BIP model of the proposed transformation. The Send/Receive BIP model corresponding to B is based on a hierarchical architecture of three layers. Figure 3 represents the Send/Receive BIP model of the BIP model given in Figure 2.

- The *Atomic Component Layer* consists of a transformation of atomic components B_i into Send/Receive atomic component B_i^{SR}. Components B_i^{SR} send asynchronously request messages to notify the scheduler layer about their current states. The bottom layer of Figure 3 includes Send/Receive atomic components $B_1^{SR}, \ldots, B_5^{SR}$.
- The *Scheduler Layer* deals with scheduling interactions. This layer consists of a set of scheduler components, each one hosting a subset of interactions. Based on requests sent by atomic components, a scheduler may decide the execution of an interaction at a given time and send back acknowledge messages to participating components specifying which transition has to be executed and when. In Figure 3, the scheduler layer consists of components S_1 and S_2.

- The *Reservation Protocol Layer* resolves conflict between schedulers. A conflict occurs when two different schedulers try to schedule two conflicting interactions. In Figure 3, The reservation protocol layer consists of component RP.

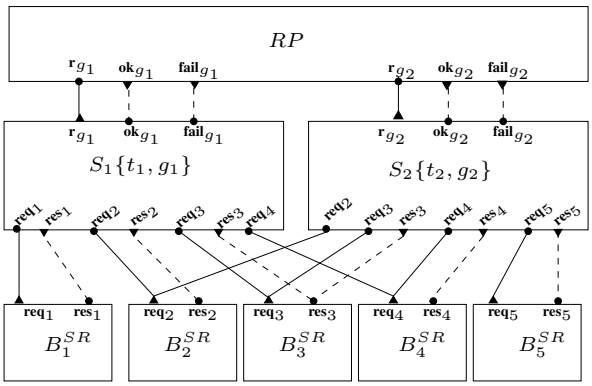

Figure 3: Send/Receive BIP Model of Figure 2.

In the Send/Receive BIP model, components rely on a common base for measuring time. We assume that they all have access to the absolute time elapsed since the system started executing. In the Send/Receive BIP model, this is represented by a single clock g shared among atomic components and schedulers. The clock g is initialized to 0 and is never reset. In the real system, g is implemented by different clocks that need to be synchronized to avoid drifts, e.g. using the Time Precision Protocol (PTP) [7]. Hence, the assumption of a single clock is valid if the difference between two clocks is always kept smaller than the precision used for representing time in the system.

The clock g is used when atomic components inform the schedulers about their timing constraints. To this end, we follow the approach of [3]: for each clock c of an atomic component B we introduce a variable ρ_c that stores the absolute time of the last reset c. If the clock c is reset by a transition of B at global time $t(g)$, we assign $t(g)$ to ρ_c. Notice that the value of c can be computed from the current value of g and ρ_c by using the equality $c = g - \rho_c$. This allows to entirely get rid of clocks of each component B, keeping only the clock g and variables ρ_c, $c \in \mathsf{C}$. Any timing constraints tc involved in a component B_i can be expressed using the clock g instead of clocks C. Using (1), we transform tc as follows: $\mathsf{tc} = \bigwedge_{c\in\mathsf{C}} l_c \leq c \leq u_c = \bigwedge_{c\in\mathsf{C}} l_c + \rho_c \leq g \leq u_c + \rho_c$. That is, tc is an interval constraint on g of the form:

$$\mathsf{tc} = \max\{l_c + \rho_c\}_{c\in\mathsf{C}} \leq g \leq \min\{u_c + \rho_c\}_{c\in\mathsf{C}}. \tag{4}$$

5.2 Transformation of Atomic Components

We transform an atomic component B into a Send/Receive atomic component B^{SR} that is capable of communicating with $scheduler$ components. To communicate, B^{SR} sends $requests$ to the schedulers that are acknowledged by $responses$. A request is sent from each location ℓ reached by B^{SR}. The request contains, for each port p, the timing constraint variable tc_p set to the timing constraint of p if the corresponding port is enabled at the current location ℓ, and set to false otherwise, the time progress condition variable tpc_B set to the time progress condition of ℓ and the *participation number* variable n which counts the number of interactions in which the component B^{SR} has participated. The value of n is used by the reservation protocol to resolve conflicts between interactions. The variables included in the request are updated whenever the component B^{SR} reaches a new state.

When the scheduler selects an interaction involving B^{SR} for execution, it notifies the component by a response containing the chosen port p^{ex} and the execution date t^{ex}.

As explained in Subsection 5.1, Send/Receive atomic component B^{SR} relies on single clock g to express timing constraints. Therefore, we include in B^{SR} a reset variable ρ_c for each clock $c \in \mathsf{C}$. Variable ρ_c is updated to t^{ex} whenever the corresponding transition of B resets clock c.

Since each request sent by a component is acknowledged by a response from the scheduler, we include transitions for sending requests and transitions for receiving responses. To this end, for each place ℓ we include two places namely $\perp_\ell^{\textbf{req}}$ and $\perp_\ell^{\textbf{res}}$. We are now ready to define the transformation from B into B^{SR}

DEFINITION 9 *Let $B = (L, P, \mathsf{C}, T, \mathsf{tpc})$ be an atomic component. The corresponding Send/Receive atomic component is $B^{SR} = (L^{SR}, P^{SR}, T^{SR}, \mathsf{C}^{SR}, \mathsf{tpc}^{SR})$ such that:*

- $L^{SR} = L^{\textbf{req}} \cup L^{\textbf{res}} \cup L$, *where* $L^{\textbf{req}} = \{\perp_\ell^{\textbf{req}} \mid \ell \in L\}$ *and* $L^{\textbf{res}} = \{\perp_\ell^{\textbf{res}} \mid \ell \in L\}$.
- $P^{SR} = P \cup \{\textbf{req}\} \cup \{\textbf{res}\}$ *where* **req** *is a* send-port *and* **res** *is a* receive-port. *The set of variables* $X_{\textbf{req}} = \{\mathsf{tc}_p\}_{p\in P} \cup \{\mathsf{tpc}_B\} \cup \{n\}$ *is associated to request port* **req**. *The set of variable* $X_{\textbf{res}} = \{p^{ex}\} \cup \{t^{ex}\}$ *is associated to response port* **res**.
- $\mathsf{C}^{SR} = \{g\}$.
- *For each place* $\ell \in L$, T^{SR} *includes a* request *transition* $\tau_\ell^{\textbf{req}} = (\perp_\ell^{\textbf{req}}, \textbf{req}, \mathtt{true}, \emptyset, \ell)$ *and a response transition* $\tau_\ell^{\textbf{res}} = (\ell, \textbf{res}, \mathtt{true}, \emptyset, \perp_\ell^{\textbf{res}})$.
- *For each transition* $\tau = (\ell, p, \mathsf{tc}, r, \ell') \in T$, T^{SR} *includes an* execution *transition* $\tau_p = (\perp_\ell^{\textbf{res}}, p, g = t^{ex}, \emptyset, \perp_{\ell'}^{\textbf{req}})$. *In addition to timing constraint* $g = t^{ex}$, τ_p *is guarded by a guard on* p^{ex} *variable,* $p^{ex} = p$. *Finally, this transition has the following update function:*

$$\begin{array}{l}\forall c \in r, \rho_c = t^{ex},\\ \mathsf{tc}_{p'} = \begin{cases} \mathsf{tc}' \text{ if } (\ell', p', \mathsf{tc}', r', \ell'') \in T \\ \mathtt{false} \text{ otherwise}\end{cases}\\ \mathsf{tpc}_B := \mathsf{tpc}_{\ell'},\\ n := n + 1.\end{array}$$

We recall that $\mathsf{tc}_{p'}$ *and* tpc_B *are expressed using clock* g *and are computed using* ρ_c *as shown in* (4).

- tpc^{SR} *is a function defined as follows:*

$$\mathsf{tpc}^{SR}(\ell) = \begin{cases} \mathsf{tpc}_\ell \text{ if } \ell \in L \cup L^{\textbf{req}} \\ g \leq t^{ex} \text{ if } \ell \in L^{\textbf{res}}.\end{cases}$$

In the above definition, the execution of a transition $\tau = (\ell, p, \mathsf{tc}, r, \ell')$ of a component B corresponds to the following three steps in B^{SR}. Firstly, a request transition $\tau_\ell^{\textbf{req}}$ transmits necessary information used by the scheduler for computing enabled interactions involving B^{SR}. Secondly, a response transition $\tau_\ell^{\textbf{res}}$ is executed once the scheduler decides to execute an interaction involving B^{SR}. The response contains the selected port p^{ex} and the chosen execution date t^{ex}. Finally, the execution transition τ_p executes the port p corresponding to p^{ex} at the chosen date t^{ex}.

Note that in B^{SR}, we put at locations $\perp_\ell^{\textbf{req}}$ and ℓ the time progress condition tpc_ℓ of location ℓ originally defined in the atomic component B. This is to ensure that B^{SR} sends its request to the scheduler and receives the response before the time progress condition tpc_ℓ becomes false. The time progress condition $g \leq t^{ex}$ of location $\perp_\ell^{\textbf{res}}$ with the timing constraint $g = t^{ex}$ of port p ensures the execution of p at the chosen date t^{ex}.

Figure 4 illustrates the transformation of the component B in Figure 1 into its corresponding Send/Receive component B^{SR}. The dashed locations represent the intermediate locations. The update functions $\mathtt{f}_{\textbf{give}}$ and $\mathtt{f}_{\textbf{take}}$ are defined as follows:

$$\mathtt{f}_{\textbf{give}} = \begin{cases} \rho_c := t^{ex} \\ \mathsf{tc}_{\textbf{take}} := g = P + \rho_c \\ \mathsf{tc}_{\textbf{give}} := \mathtt{false} \\ \mathsf{tpc}_B := g \leq P + \rho_c \\ n := n + 1\end{cases}$$

$$\mathtt{f}_{\textbf{take}} = \begin{cases} \rho_c := t^{ex} \\ \mathsf{tc}_{\textbf{take}} := \mathtt{false} \\ \mathsf{tc}_{\textbf{give}} := E + \rho_c \leq g \leq D + \rho_c \\ \mathsf{tpc}_B := g \leq D + \rho_c \\ n := n + 1\end{cases}$$

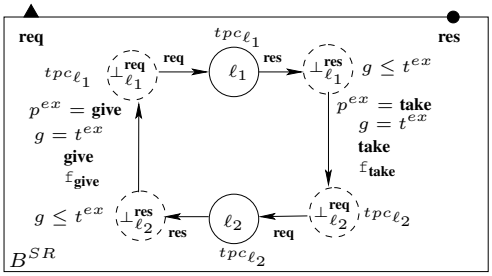

Figure 4: Transformation of atomic component of Figure 1.

5.3 Building Distributed Scheduler

In this subsection, we describe how to build a distributed scheduler component. Consider a BIP model $\gamma(B_1 \cdots B_n)$ and a partition of the set of interactions $\{\gamma_j\}_{j=1}^m$ (*classes* of interactions γ_j are disjoint and cover all the interactions of γ). Each class of interaction γ_j is handled by a single scheduler component S_j. The partition $\{\gamma_j\}_{j=1}^m$ is a parameter of our method that can be used to optimize the generated implementations. It also determines whether or not a conflict between interactions can be resolved locally. Consider conflicting interactions $a \in \gamma_j$ and $b \in \gamma_k$. We distinguish between two types of conflict for a and b, according to the partition $\{\gamma_j\}_{j=1}^m$.

- A conflict is *internal* if a and b belong to the same class of the partition, i.e. $j = k$. In this case, it can be resolved by the scheduler component S_j responsible for a and b.
- A conflict is *external* if a and b belong to different classes of the partition, i.e. $j \neq k$. External conflicts cannot be resolved by schedulers alone, and are referred to the reservation protocol layer. The scheduler S_j sends a request to the reservation protocol to reserve an interaction and receives a response by either **ok** if the reservation succeeds or **fail** if the reservation cannot be granted.

The scheduler component S_j receives request messages sent by the Send/Receive atomic components. Based on the request message received, S_j component calculates the set of enabled interactions and their timing constraints and selects one of them for execution (either locally or by means of the reservation protocol layer). It chooses also a date for the execution of the selected interaction. Then, it sends a response to each component participating in the chosen interaction. The response contains the port and the date for the execution. We define scheduler components using Petri nets as they provide a compact format for the description of the behavior of concurrent systems.

DEFINITION 10 *A* Petri net *is defined by a triple $S = (L, P, T)$, where L is a set of* places*, P is a set of ports, and $T \subseteq 2^L \times P \times 2^L$ is a set of transitions. A transition τ is a triple $({}^\bullet\tau, p, \tau^\bullet)$, where ${}^\bullet\tau$ is the set of* input places *of τ and $\tau^\bullet$ is the set of* output places *of τ.*

A Petri net is often modeled as a directed bipartite graph $G = (L \cup T, E)$. Places are represented by circular vertices and transitions are represented by rectangular vertices (see Figure 5). The set of directed edges E is the union of the sets $\{(\ell, \tau) \in L \times T \mid \ell \in {}^\bullet\tau\}$ and $\{(\tau, \ell) \in T \times L \mid \ell \in \tau^\bullet\}$. We depict the *state* of a Petri net by marking its places with tokens [8]. We say that a place is *marked* if it contains a token. A transition τ is *enabled* at a state if all its input places ${}^\bullet\tau$ are marked. Upon the execution of τ, tokens of input places ${}^\bullet\tau$ are removed and tokens in output places in $\tau^\bullet$ are added.

Given an initial state $m_0 \subseteq L$, a Petri net (L, P, T) is *1-Safe* if for any execution from m_0 output places of enabled transitions are never marked. The behavior of a 1-Safe Petri net (L, P, T) is defined as a finite labeled transition system $(2^L, P, \rightarrow)$, where 2^L is the set of states, P is the set of labels, and $\rightarrow \subseteq 2^L \times P \times 2^L$ is the set of transitions defined as follows. We have $(m, p, m') \in \rightarrow$, denoted by $m \xrightarrow{p} m'$, if there exists $\tau = ({}^\bullet\tau, p, \tau^\bullet) \in T$ such that ${}^\bullet\tau \subseteq m$ and $m' = (m \backslash {}^\bullet\tau) \cup \tau^\bullet$. In this case, we say that

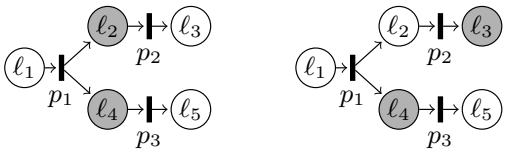

Figure 5: A simple Petri net

p is *enabled* at m. We say that the Petri net (L, P, T) is *deterministic* if for any execution from m_0 two transitions $\tau_1 \neq \tau_2$ labeled by same port p are not enabled at the state.

Consider a scheduler S_j handling the subset of interactions γ_j. Let $a = \{p_i \mid i \in I\}$ be an interaction belonging to γ_j. Scheduling a needs receiving requests from components B_i participating in a in (i.e. $B_i \in part(a)$ and components B_j oberved by a (i.e. $B_j \in obs(a)$). Only requests from the participants need to be acknowledged by a response whenever the interaction a is cheduled. Regarding requests from observed components, they are used to compute a safe date for scheduling a. Note that only time progress conditions are used from requests received from observed components.

In order to choose a safe date, S_j computes the safe timing constraint $\mathsf{safe\text{-}tc}_a$ of a that corresponds to the conjunction of timing constraints and time progress conditions of components participating in a with the time progress conditions of components observed by a:

$$\mathsf{safe\text{-}tc}_a = \bigwedge_{B_i \in part(a)} [\mathsf{tc}_{p_i} \wedge \mathsf{tpc}_{B_i}] \bigwedge_{B_j \in obs(a)} \mathsf{tpc}_{B_j} \tag{5}$$

Given a scheduling policy $\mathcal{P}$, the safe timing constraint $\mathsf{safe\text{-}tc}_a$ and the actual valuation of clock g, S_j computes the set of safe dates $\mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a)$ and schedules interaction a only if this set is not empty. In fact, it chooses a date from this set and sends it to the participating components. In the case where interaction a is externally conflict, the scheduler starts the reservation mechanism if the set $\mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a)$ is not empty. If the reservation protocol responds by **ok**, the scheduler checks again the non-emptiness of $\mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a)$ to ensure finding a safe date for scheduling interaction a. In the case where the set $\mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a)$ is not empty, the scheduler proceeds to schedule interaction a. In the other case, we propose to report an error and stop the execution as this situation is inconsistent in the system (reservation protocol confirms the execution of a, whereas the scheduler is not able to execute it). In practice, this situation can occur when the communication delays between the reservation protocol and the scheduler are too long, which may invalidate the timing constraint of a. A solution this problem could be to integrate a cancel mechanism between the scheduler and the reservation protocol. In fact, the scheduler may send a cancel request to the reservation protocol so as to inform it that the interaction a will not be executed. In this paper, we do not detail this mechanism.

The Petri net that defines the behavior of scheduler S_j handling the subset of interaction γ_j is constructed as follows.

Variables. The set of variables is the following.

- We include variables updated whenever a request from component B_i participating in or observed by an interaction of γ_j is received. They consist of the timing constraint variable tc_p for each port p of B_i, the time progress condition variable tpc_{B_i} and the participation number variable n_i. Recall that for observed components B_i, only variable tpc_{B_i} is used by the scheduler.
- We include also variables updated whenever interaction $a \in \gamma_j$ is scheduled. They consist of the execution date variable t_a^{ex}, the port execution variable p_i^{ex} and the execution date variable t_i^{ex} for each component B_i participating in a.

Clocks. The set of clocks contains the clock g.
Places. The set of places is the following.

- For each component B_i participating in an interaction of γ_j, we include $waiting$, $received$ and $sending$ places w_i, r_i and s_i respectively. Place w_i has time progress condition defined by tpc_{B_i}. Place s_i has the time pogress condition $g \leq t_i^{ex}$.

- For each interaction $a \in \gamma_j$ that is in external conflict, we include an $engaged$ place e_a. This place has the time progress condition defined by $\bigwedge_{B_i \in part(a)} \mathsf{tpc}_{B_i}$.

Ports. The set of ports is the following.

- For each component B_i, we include a request port $\mathbf{req}_i$ and a response port $\mathbf{res}_i$.
- For each interaction $a \in \gamma_j$ that is not in external conflict, we include a unary port $\mathbf{sched}_a$.
- For each interaction $a \in \gamma_j$ that is in external conflict, we include reservation ports $\mathbf{r}_a$, $\mathbf{ok}_a$ and $\mathbf{fail}_a$. We associate to port $\mathbf{r}_a$ the set of variables $\{n_i\}_{B_i \in part(a)}$.

Transitions. The set of transitions is the following.

- In order to receive requests from a component B_i^{SR}, we include a request transition $(w_i, \mathbf{req}_i, r_i)$. We include for each externally conflicting interaction a transition $(r_i, \mathbf{req}_i, r_i)$ and $(e_a, \mathbf{req}_i, e_a)$, where i is the index of component B_i involved in a, to receive new requests when B_i takes part in other conflicting interaction.
- In order to send response to component B_i, we include transition $(s_i, \mathbf{res}_i, w_i)$.
- In order to schedule an interaction $a = \{p_i\}_{i \in I} \in \gamma_j$ that is not in external conflict, we include transition $\tau_{\mathbf{sched}_a} = (\{r_i \mid B_i \in part(a) \cup obs(a)\}, \mathbf{sched}_a, \{s_i \mid B_i \in part(a)\} \cup \{r_i \mid B_i \in obs(a)\})$ guarded by $\mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a) \neq \emptyset$. This transition selects a safe date $t_a^{ex} \in \mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a)$ and updates variables p_i^{ex} to p_i and variables t_i^{ex} to t_a^{ex}.
- In order to request reservation of an interaction $a = \{p_i\}_{i \in I} \in \gamma_j$ that is in external conflict, we include a requesting reservation transition $\tau_{\mathbf{r}_a} = (\{\mathbf{r}_i \mid B_i \in part(a) \cup obs(a)\}, r_a, \{e_a\} \cup \{\mathbf{r}_i \mid B_i \in obs(a)\})$ guarded by $\mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a) \neq \emptyset$.
- For the case where the reservation protocol responds positively, we include the transition $\tau_{\mathbf{ok}_a} = (\{e_a\}, \mathbf{ok}_a, \{s_i \mid B_i \in part(a)\})$ guarded $\mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a) \neq \emptyset$. This transition selects a safe date $t_a^{ex} \in \mathcal{P}(t(g), \mathsf{safe\text{-}tc}_a)$ and updates variables p_i^{ex} to p_i and variables t_i^{ex} to t_a^{ex}.
- For the case where the reservation protocol responds negatively, we include the transition $\tau_{\mathbf{fail}_a} = (\{e_a\}, \mathbf{fail}_a, \{r_i \mid B_i \in part(a)\})$.

Note that the time progress condition of received place r_i corresponding to component B_i is defined by tpc_{B_i} variable which contains the current time progress condition of component B_i. This time progress condition enforces the scheduler to schedule an interaction involving component B_i before that its current time progress condition becomes false. The time progress condition of place e_a, where a is an externally conflicting interaction, is defined by $\bigwedge_{B_i \in part(a)} \mathsf{tpc}_{B_i}$. This time progress condition enforces receiving a response from the reservation protocol before that one of time progress conditions of components participating in a becomes false. Finally, the time progress condition of place s_i corresponding to component B_i is defined by $g \leq t_i^{ex}$, where t_i^{ex} is the date at which the component B_i should execute. This time progress condition ensures that the response is sent before t_i^{ex}.

Figure 6 shows the scheduler component S_1 of Figure 3. For sake of readability, all time progress conditions, guards and update functions are not shown in the figure. S_1 handles interactions t_1 and g_1. As t_1 is not conflicting with any other interaction, it is handled locally in S_1. However, g_1 is in external conflict with g_2. Its scheduling requires requesting reservation from the reservation protocol layer through port $\mathbf{r}_{g_1}$. Moreover, to compute interaction g_1, S_1 has to receive requests from participating components B_2 and B_3 as well as requests from observed component B_4. Only participating components B_2 and B_3 are notified for the execution of g_1.

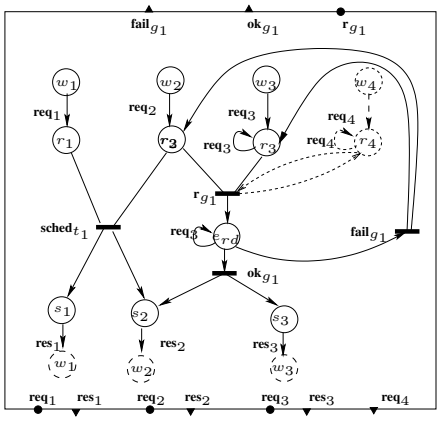

Figure 6: Decentralized Scheduler S_2 of Figure 3.

5.4 Reservation Protocol

The reservation protocol layer implements an algorithm that solves the committee coordination problem. The adopted algorithm is based on the idea of message-count technique presented in [9]. This technique is based on counting the number of times that a component participates in an interaction. Conflicts are resolved by ensuring that each participation number is used only once. In order to implement the algorithm, the reservation protocol keeps variables N_i which store the last value of the participation number of each component B_i. Whenever a reserve request $\mathbf{r}_a$ for interaction a is received, the message provides the set of participation numbers $\{n_i\}_{B_i \in part(a)}$. If for each component B_i the participation number n_i is greater than N_i, then the reservation protocol acknowledges successful through port $\mathbf{ok}_a$ and updates N_i to the values of n_i. On the contrary, if there exists a component B_i whose participation number n_i is less or equal to what the reservation protocol has recorded, then the corresponding component has already participated in an interaction with this participation number and the reservation protocol replies failure via port $\mathbf{fail}_a$. In this paper, we consider a centralized implementation of the reservation protocol. In this implementation, there is one centralized component that implements the protocol described above, and constructed as follows (see Figure 7).

- For each component B_i, we include variable N_i and for each interaction a we include variable n_i^a.
- For each interaction a handled by the reservation protocol, we include two places w_a and r_a, three ports $\mathbf{r}_a$, $\mathbf{ok}_a$ and $\mathbf{fail}_a$, and three transitions $\tau_{\mathbf{r}_a} = (w_a, \mathbf{r}_a, r_a)$, $\tau_{\mathbf{ok}_a} = (r_a, \mathbf{ok}_a, w_a)$ and $\tau_{\mathbf{fail}_a} = (r_a, \mathbf{fail}_a, w_a)$. The receive-port $\mathbf{r}_a$ receives reservation requests containing fresh values of variables n_i^a. The send-ports $\mathbf{ok}_a$ and $\mathbf{fail}_a$ accept or reject the latest reservation request, and N_i variables are updated in case of positive response.

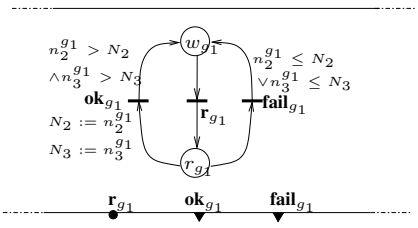

Figure 7: Fragment of the centralized reservation protocol RP of Figure 3 handling interaction g_1.

Two other implementations of the reservation protocol are presented in [10], namely token ring based implementation, and dining philosophers based implementation. These implementations are more distributed as they consider one reservation protocol component per externally conflicting interaction. These implementations are also considered by our method, but not presented in this paper.

5.5 Send/Receive Interactions

In this subsection, we define the Send/Receive interactions between the components defined thus far. Given a BIP model $B = \gamma(B_1, ..., B_n)$ and a partition $\{\gamma_j\}_{j=1}^{m}$, we obtain a Send/Receive BIP model $B_{CP}^{SR} = \gamma^{SR}(B_1^{SR}, ..., B_n^{SR}, S_1, ..., S_m, RP)$. The set of Send/Receive interactions γ^{SR} is constructed as follows:

- For each component B_i^{SR}, let $S_{j_1}, ..., S_{j_l}$ be the scheduler components handling interactions involving B_i^{SR}. We include in γ^{SR} the request interaction $(B_i^{SR}.\mathbf{req}, S_{j_1}.\mathbf{req}_i, ..., S_{j_l}.\mathbf{req}_i)$.
- For each scheduler component S_j and for each component B_i^{SR} participating in an interaction handled by S_j, we include in γ^{SR} the response interaction $(S_j.\mathbf{res}_i, B_i^{SR}.\mathbf{res}_i)$.
- For each externally conflicting interaction a handled by S_j, we add in γ^{SR} the interaction $(S_j.\mathbf{r}_a, RP.\mathbf{r}_a)$. Likewise, we include interactions $(RP.\mathbf{ok}_a, S_j.\mathbf{ok}_a)$ and $(RP.\mathbf{fail}_a, S_j.\mathbf{fail}_a)$.

LEMMA 1 *The Send/Receive model B_{CP}^{SR} meets the properties of Definition 8.*

Proof 3 *The first two constraints of Definition 8 are trivially met by construction. We now prove that the third constraint also holds; i.e, whenever a send-port is enabled, all its associated receive-ports are enabled as well.*

- *Between a Send/Receive component B_i^{SR} and a scheduler S_j, we consider the places w_i, r_i and s_i. If there is no token in s_i place, then, it is easy to see that from this configuration, only interactions $\mathbf{req}_i$ is enabled. If there is a token at place s_i, it results from the execution of transition $\mathbf{sched}_a$ or $\mathbf{ok}_a$ in the scheduler. In the first case, a is internally or not conflicting interaction. In this case, there is no other interaction than a that could be scheduled and could activate place s_i. In the second case, the interaction a is externally conflicting and is refereed to the reservation protocol. The latter uses the current participation number of B_i for the execution of a and no other interaction is granted using the same participation number. Thus, in both cases, s_i is the only active place from which a notification could be sent.*
- *Between the scheduler S_j and the reservation protocol, we consider the places e_a in the scheduler S_j, w_a and r_a in the reservation protocol. If e_a is empty, and w_a is active, only reservation request through port $\mathbf{r}_a$ is enabled. When the $\mathbf{r}_a$ request is sent, the place e_a and r_a become active. From this configuration, only send ports $\mathbf{ok}_a$ and $\mathbf{fail}_a$ are enabled in the conflict resolution protocol, and the associated ports are also enabled in the scheduler.*

6 Experimental Results

In this section, we present the results of our experiments. Our implementation automatically generates C++ code from the Send/Receive BIP model developed in Section 5, where Send/Receive interactions are implemented by TCP sockets primitives. Code generation involves generating stand-alone executables for each component in each layer of the Send/Receive BIP model. The code of each component simulates its automaton or Petri net using the technique presented in [11]. During the execution, the components send, receive messages, or do internal computations. Note that the execution stops if the time progress conditions of atomic components are not met. This can occur when the platform, on which the Send/Receive BIP model is implemented, is not fast enough to meet the time progress conditions. To check the implementability of the system on a given platform along with its timing constraints, one may derive a physical model from the Send/Receive BIP model by introducing time delays of transitions [3]. In this paper, we do not discuss this transformation.

We conduct experiments on an application consisting on a simulation of a set of robots collaborating to perform a given task. The scenario is described as follows. Initially, the robots are randomly distributed over an arena. They start by exploring the arena in order to find each others. When 3 robots become sufficiently close, they group themselves and go towards an object and push it.

Figure 8 shows the model of a single robot. We use timing constraints and time progress conditions to express a periodic sensors reading (P=200ms). Notice that such a component has non-decreasing deadlines

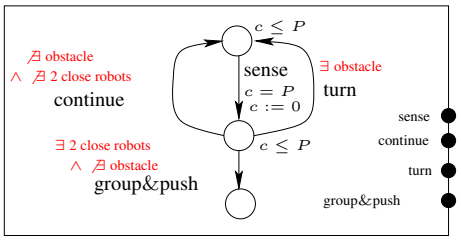

Figure 8: Model of a single robot.

since it satisfies the conditions of Proposition 1. Note that transitions turn, continue and group&push have mutually exclusive guards (they cannot be enabled at the same time).

The BIP model of the application consists of N robots. The "group&push" action is modeled by an interaction that synchronizes group&push transitions of any 3 robots, enabled only if there are 3 robots which are sufficiently close to each others. We denote by g_k the "group&push" interaction between 3 robots where k is the robots group identifier. These interactions are pairwise conflicting as for any two interactions there is at least one shared ports. Note that for each robot, there are 3 unary interactions which are sense_i, continue_i and turn_i.

Using the transformations described in Section 5, we transform the BIP model of our application into Send/Receive BIP model where all unary interactions of robot R_i are handled by a single scheduler S_i and each g_k interaction of group k is handled by a single scheduler S_k.

In our example, each two interactions g_k and $\text{g}_{k'}$ are in conflict. Therefore, each g_k interaction involves its participant components and observes all remaining components. The scheduler S_k is then required to receive requests from all robots: 3 robots participating in the interaction and $N - 3$ other observed.

Our method to optimize the number of observed components, described in Section 4, allows removing all observed components for each g_k interaction, as the predicate $\mathsf{reduce}_{g_k}(R_j)$ holds or each $R_j \in obs(g_k)$. This could be explained as follows. In the BIP model, all the robots have the same behavior and the same period for sensing which makes them having the same deadlines when searching for each others. Thus, when 3 robots try to group themselves, they have to do it before the expiration of their periods of sensing, which are the same for the other (observed) robots. Therefore, none of the observed components will be blocked if any 3 robots group themselves.

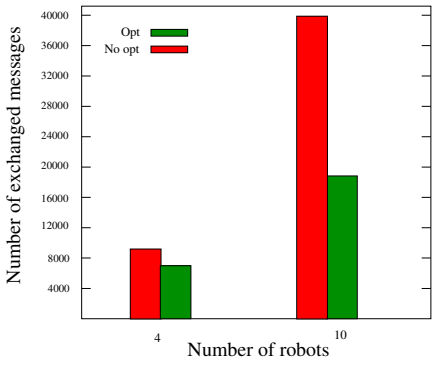

Figure 9: Number of exchanged messages needed for the execution of the application during 10s.

We measure the number of exchanged messages needed or the execution of the application during 10s. Figure 9 shows the number of exchanged messages needed or the execution of the application with 4 and 10 robots. We remark that the performances are improved in the optimized version especially for the application with 10 robots. This is because in the unoptimized version with 10 robots, the scheduler S_k

handling g_k interaction has to receive messages from all robots: 3 participating in the interaction and 7 other observed. In the optimized version, S_k receives only messages from the participant robots, which reduces drastically the number of exchanged messages compared to the unoptimized version.

7 Related Work

LOTOS [12] is a specification language based on process algebra, that encompasses multiparty interactions. In [13], the authors describe a method of executing a LOTOS specification in a distributed fashion. This implementation is obtained by constructing a tree at runtime. The root is the main connector of the LOTOS specification and its children are the subprocesses that are connected. A synchronization between two processes is handled by their common ancestor. Another framework that offers automatic distributed code generation is described in [14]. The input model consists of composition of I/O automata, from which a Java implementation using MPI for communication is generated. The model, as well as the implementation, can interact with the environment. However, connections between I/O automata (binary synchronization) are less expressive than BIP interactions, as proved in [15]. Finally, the framework in [14] requires the designer to specify low-level elements of a distributed system such as channels and schedulers.

OASIS-D [16] is an extension of OASIS approach towards distributed architectures. An OASIS application is composed of a set of real-time tasks called agents communicating through temporal variables and messages exchange. Unlike BIP approach, OASIS relies on fully deterministic behavior of the application. OASIS-D provides tool chain that (1) computes network feasibility of the application and (2) generates run-time network including the structure of the application as well as a deterministic TDMA scheduler for the network access. Our method is more general compared to OASIS-D as we generate schedulers that allow non-determinism.

PTIDES [17] is a data-flow approach for modeling event-triggered distributed real-time systems. The timed semantics of PTIDES specifies the interaction between the program and the environment based on two main assumptions which are bounded clock synchronization and bounded latencies for networks. In [17] the authors describe analysis techniques that check system implementations for satisfaction of PTIDES temporal semantics. Our framework is more expressive compared to PTIDES as we support multi-party interactions.

The closest works to this paper are the approaches in [4] and [18]. The technique in [4] transforms a BIP model into a parallel time-aware code. The main difference is unlike our approach, the method in [4] augments the code with only one centralized scheduler. Such a scheduler can potentially become a bottleneck and consequently make the generated code inefficient. The solution presented in [18] proposes to decentralize the scheduler by building a set of conflict-free schedulers. Such a solution is not the optimal choice, since one may end up with a centralized scheduler if the BIP model has a chain of conflicting interactions. Our method is more a general since it is parametrized by a conflict resolution protocol.

8 Conclusion

We presented a fully automated method for distributed implementation of BIP models consisting of a set of atomic components communicating through multiparty interactions. Each atomic component is constrained by a set of local timing constraints. We considered models that have non-decreasing deadlines that is, executing transitions cannot decrease the actual deadline of a component. Based on this property, we provide safe condition for scheduling interactions concurrently. We show that scheduling an interaction safely requires observing additional components in addition to the ones participating in the interaction. We provided a method for optimizing the number of observed components based on the use of static analysis techniques and verification methods.

Our method for automatically generating distributed implementation of BIP models consists of two transformations. The first transformation takes a BIP model as input and generates a Send/Receive BIP model in which components communicate through asynchronous message passing. The Send/Receive BIP model is composed of Send/Receive components, scheduler components, each one being responsible for

scheduling a subset of interactions, and a reservation protocol component that resolves conflict between schedulers.

We conducted experiments on an application consisting on a simulation of a set of collaborating robots implemented using our transformation method. We used our method for optimizing observed components and we show net improvement of the application for the optimized version in terms of number of exchanged messages.

For future work, we plan to pursue several directions. First, we are working on extending our method by considering more general models where deadlines may decrease when executing transitions. An other important research direction is to handle clocks drift issues in distributed real-time systems where clocks synchronization can not be assumed.

References

[1] Basu, A., Bidinger, P., Bozga, M., Sifakis, J.: Distributed semantics and implementation for systems with interaction and priority. In: FORTE. (2008) 116–133 1, 2

[2] Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126**(2) (1994) 183–235 1, 2.1, 2.1

[3] Abdellatif, T., Combaz, J., Sifakis, J.: Model-based implementation of real-time applications. In Carloni, L.P., Tripakis, S., eds.: EMSOFT, ACM (2010) 229–238 1, 5.1, 6

[4] Triki, A., Combaz, J., Bensalem, S., Sifakis, J.: Model-based implementation of parallel real-time systems. In: FASE. (2013) 235–249 1, 7

[5] Astefanoaei, L., Rayana, S.B., Bensalem, S., Bozga, M., Combaz, J.: Compositional invariant generation for timed systems. In: TACAS. (2014) 263–278 1, 4

[6] Tripakis, S.: Verifying progress in timed systems. In Katoen, J.P., ed.: ARTS. Volume 1601 of Lecture Notes in Computer Science., Springer (1999) 299–314 2.1

[7] Eidson, J.C.: Measurement, Control and Communication Using IEEE 1588. Springer (2006) 5.1

[8] Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (apr 1989) 541 –580 5.3

[9] Bagrodia, R., ed.: Process synchronization: design and performance evaluation of distributed algorithms. In Bagrodia, R., ed.: TSE, IEEE (1989) 5.4

[10] Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: From high-level component-based models to distributed implementations. In: EMSOFT. (2010) 209–218 5.4

[11] Triki, A., Bonakdarpoor, B., Combaz, J., Bensalem, S.: Automated conflict-free concurrent implementation of timed component-based models. Technical report, Verimag Research Report 6

[12] ISO/IEC: Information Processing Systems – Open Systems Interconnection: LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behavior. (1989) 7

[13] von Bochmann, G., Gao, Q., Wu, C.: On the distributed implementation of lotos. In: FORTE. (1989) 133–146 7

[14] Tauber, J.A., Lynch, N.A., Tsai, M.J.: Compiling IOA without global synchronization. In: Symposium on Network Computing and Applications (NCA). (2004) 121–130 7

[15] Bliudze, S., Sifakis, J.: A notion of glue expressiveness for component-based systems. In: Concurrency Theory (CONCUR). (2008) 508–522 7

[16] Faucou, S., Burns, A., 0001, L.G., eds.: Scheduling safety-critical real-time bus accesses using Time-Constrained Automata. In Faucou, S., Burns, A., 0001, L.G., eds.: RTNS. (2011) 7

[17] Eidson, J., Lee, E.A., Matic, S., Seshia, S.A., Zou, J.: Distributed real-time software for cyber-physical systems. Proceedings of the IEEE **100** (2012) 45 – 59 7

[18] Triki, A., Bonakdarpour, B., Combaz, J., Bensalem, S.: Automated conflict-free concurrent implementation of timed component-based models. In: NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings. (2015) 359–374 7