Rigorous System Design Flow for Autonomous Systems

Saddek Bensalem, Marius Bozga, Jacques Combaz, Ahlem Triki

Rigorous System Design Flow for Autonomous Systems

Saddek Bensalem, Marius Bozga, Jacques Combaz, and Ahlem Triki

Verimag, France
{firstname.lastname}@imag.fr

Abstract. We currently lack rigorous approaches for modeling and implementing complex systems. BIP (Behavior, Interaction, Priority) is a component-based framework intended to rigorous system design. It relies on single semantic model for system descriptions all along the design flow. It also includes methods and tools for guaranteeing system correctness to avoid a posteriori verification. Our approach is to check safety properties (e.g. deadlock freedom) at design time using D-Finder verification tool. In addition, source-to-source transformers allow progressive refinement of the application to generate a correct implementation. Our framework was successfully applied in various context including robotics case studies presented here.

1 Introduction

System design is the process leading to a mixed hardware/software system meeting given specifications. It involves the development of application software taking into account features of an execution platform. The latter is defined by its architecture involving a set of processors equipped with hardware-dependent software such as operating systems as well as primitives for coordination of the computation and interaction with the external environment.

System design radically differs from pure software design in that it must take into account not only functional but also extra-functional specifications regarding the use of resources of the execution platform such as time, memory and energy. Meeting extra-functional specifications is essential for the design of embedded systems. It requires evaluation of the impact of design choices on the overall behavior of the system.

We currently lack rigorous techniques for deriving global models of a given system from models of its software and its execution platform. We call rigorous a design flow which allows guaranteeing essential system properties. Most of the existing rigorous design flows privilege a unique programming model together with an associated compilation chain adapted for a given execution model. For example, synchronous system design relies on synchronous programming models and usually targets hardware or sequential implementations on single processors [1]. Alternatively, real-time programming based on scheduling theory for periodic tasks, targets dedicated real-time multitasking platforms [2].

We strongly believe that a rigorous design flow should be *model-based*, that is, all the system description should be based on a single semantic model, should be *component-based*, that is, it provides primitives for building composite components as the composition of simpler component, and should rely on tractable theory for guaranteeing *correctness by construction* to avoid as much as possible monolithic a posteriori verification. An instance of rigorous design flow is the BIP approach presented below.

The BIP Desgin Flow. Behavior—Interaction—Priority (BIP) is a component framework intended to rigorous system design. It allows the construction of composite hierarchically structured components from atomic components characterized by their behavior and their interface. Components are composed by layered application of interactions and of priorities. Interactions express synchronization constraints between actions of the composed components while priorities are used to filter amongst possible interactions and to steer system evolution so as to meet performance requirements e.g. to express scheduling policies. Interactions are described in BIP as the combination of two types of protocols: rendez-vous to express strong symmetric synchronization and broadcast to express triggered asymmetric synchronization. The combination of interactions and priorities confers BIP expressiveness not matched by any other existing formalism [3]. It defines a clean and abstract concept of architecture separate from behavior. Architecture in BIP is a first class concept with well-defined semantics that can be analyzed and transformed. BIP relies on rigorous operational semantics that has been implemented by three Execution Engines for centralized, distributed and real-time execution. It is used as a unifying semantic model in a rigorous system design flow. Rigorousness is ensured by two kinds of tools: 1) D-Finder a verification tool for checking safety properties and deadlock-freedom in particular; 2) source-to-source transformers that allow progressive refinement of the application to get a correct implementation.

BIP can be considered as an ADL (Architecture Description Language) or as a coordination language as it focuses on the organization of computation between components. As other existing ADL such as ACME [4] and Darwin [5], BIP uses the concept of connector to express coordination between components. Nonetheless, connectors in BIP are stateless. There is a clear distinction between architecture which involves connectors and priorities and behavior. Another significant difference is that BIP is intended to system modeling as it directly encompasses timing and resource management aspects. It differs from other system modeling formalisms which either seek generality at the detriment of rigorousness, such as SySML [6] and AADL [7] or have a limited scope as they are based on specific models of computation such as Ptolemy [8].

In previous work, we successfully applied the BIP design flow to the robot DALA, an autonomous rover for extraterrestria exploration [9–11]. This paper is based on the extension of BIP to time proposed in [12], which was not considered by [9–11]. Its contributions[1] are: *(i)* the application of recently developed

[1] This work was supported by the European Integrated Project 257414 ASCENS.

verification and validation techniques to autonomous systems case studies, and *(ii)* the extension of the method for generation of distributed implementations proposed in [13] to timed systems. The rest of the paper is organized as follows. Section 2 provides a formalization of the BIP language and its semantics. Section 3 describes the BIP toolchain consisting mainly of: a compiler, including backends for the generation of both single-threaded and multi-threaded C++ code, as well as message passing based implementations for their deployment on distributed platforms, and verification and validation tools for checking the correctness of the system and its performance. Finally, Section 4 demonstrates our approach by the application of our tools to various robotics case studies.

2 Basic Semantic Model of BIP

Definition 1 (abstract model). *An* abstract model *is a timed automaton* $M = (\mathsf{A}, \mathsf{Q}, \mathsf{X}, \longrightarrow, \mathsf{tpc})$ *such that:*

- A *is a finite set of* actions.
- Q *is a finite set of* control locations
- X *is a finite set of* clocks
- $\longrightarrow$ *is a finite set of labeled transitions. A transition is a tuple* (q, a, g, r, q') *where* $q, q' \in \mathsf{Q}$ *are control locations,* a *is an action executed by the transition,* g *a constraint over* X *called* guard*, and* r *is a subset of clocks that are reset by the transition. We write* $q \xrightarrow{a,g,r} q'$ *for* $(q, a, g, r, q') \in \longrightarrow$
- tpc *is a function associating to each control location* $q \in \mathsf{Q}$ *a constraint* $\mathsf{tpc}[q]$ *over* X *called* time progress condition.

An abstract model describes the platform-independent behavior of the system. Timing constraints, that is, guards of transitions and time progress conditions of control locations, are any boolean combination of simple constraints of the form $x \sim k$, where $x \in \mathsf{X}$ is a clock, $k \in \mathbb{N}$ is a non-negative integer, and $\sim$ is a comparison operator: $\sim \in \{<, \leq, \geq, >\}$. They take into account only user requirements (e.g. deadlines, periodicity, etc.). The semantics assumes timeless execution of actions.

Definition 2 (abstract model semantics). *An abstract model* $M = (\mathsf{A}, \mathsf{Q}, \mathsf{X}, \longrightarrow)$ *defines a transition system* TS*. States of* TS *are pairs* (q, v)*, where* q *is a control location of* M *and* $v : \mathsf{X} \to \mathbb{R}^+$ *is a valuation of the clocks* X *mapping each clock* $x \in \mathsf{X}$ *to its current value* $v(x) \in \mathbb{R}^+$*, where* $\mathbb{R}^+$ *denotes the set of non-negative reals.*

- Actions. *We have* $(q, v) \xrightarrow{a} (q', v[r \mapsto 0])$ *if* $q \xrightarrow{a,g,r} q'$ *in* M *and both* $g(v)$ *and* $\mathsf{tpc}[q'](v[r \mapsto 0])$ *are true, where* $v[r \mapsto 0]$ *denotes the valuation of the clocks such that* $v[r \mapsto 0](x) = 0$ *if* $x \in r$*,* $v[r \mapsto 0](x) = v(x)$ *otherwise.*
- Time steps. *For a* waiting time $\delta \in \mathbb{R}^+$*,* $\delta > 0$*, we have* $(q, v) \xrightarrow{\delta} (q, v + \delta)$ *if the time progress condition* $\mathsf{tpc}[q]$ *allows the system to wait for* δ *at* (q, v)*, that is, if for all* $\delta' \in [0, \delta]$*,* $\mathsf{tpc}[q](v + \delta')$ *is true.*

In an abstract model, clocks are non-negative real variables increasing synchronously. Guards are used to specify for which values of the clocks the actions may take place, and time progress conditions specify whether the system can wait at a given state or needs to execute an action to leave this state. Given an abstract model $M = (\mathsf{A}, \mathsf{Q}, \mathsf{X}, \longrightarrow)$, an *execution sequence* of M from an *initial* state (q_0, v_0) is a maximal sequence actions and time-steps $(q_i, v_i) \xrightarrow{\sigma_i} (q_{i+1}, v_{i+1})$, $\sigma_i \in \mathsf{A} \cup \mathbb{R}^+$, $i \geq 0$.

Example 1. Consider an abstract model $M = (\mathsf{A}, \mathsf{Q}, \{x\}, \longrightarrow)$ with two actions $\mathsf{A} = \{sync_1, p\}$, two states $\mathsf{Q} = \{q^1, q^2\}$, a single clock x, and two transitions $\longrightarrow = \{\ (q^1, sync_1, \emptyset, \{x\}, q^2), (q^2, p, [10 \leq x \leq 20]^{\mathsf{d}}, \emptyset, q^1)\}$ (see Figure 1). It can be easily shown that the execution sequences of M from the initial state $(q^2, 0)$ that are an infinite repetition of the sequence $(q^2, 0) \xrightarrow{\delta_1} (q^2, \delta_1) \xrightarrow{p} (q^1, \delta_1) \xrightarrow{\delta_2} (q^1, \delta_1 + \delta_2) \xrightarrow{sync_1} (q^2, 0)$, where $10 \leq \delta_1 \leq 20$.

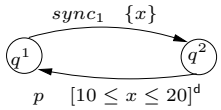


Fig. 1. Example of abstract model

Definition 3 (composition of abstract models). *Let $M_i = (\mathsf{A}_i, \mathsf{Q}_i, \mathsf{X}_i, \longrightarrow_i, \mathsf{tpc}_i)$, $1 \leq i \leq n$, be a set of abstract models. We assume that their sets of actions and clocks are disjoint, i.e. for all $i \neq j$ we have $\mathsf{A}_i \cap \mathsf{A}_j = \emptyset$ and $\mathsf{X}_i \cap \mathsf{X}_j = \emptyset$. A set of* interactions *γ is a subset of 2^{A}, where $\mathsf{A} = \bigcup_{i=1}^{n} \mathsf{A}_i$, such that any interaction $a \in \gamma$ contains at most one action of each component M_i, that is, $a = \{\ a_i \mid i \in I\ \}$ where $a_i \in \mathsf{A}_i$ and $I \subseteq \{\ 1, 2, \ldots, n\ \}$. The* composition *of the abstract models M_i, $1 \leq i \leq n$, by using a set of interactions γ, denoted by $\gamma(M_1, \ldots, M_n)$, is the composite abstract model $M = (\gamma, \mathsf{Q}, \mathsf{X}, \longrightarrow_\gamma, \mathsf{tpc})$ such that:*

- $\mathsf{Q} = \mathsf{Q}_1 \times \mathsf{Q}_2 \times \ldots \times \mathsf{Q}_n$
- $\mathsf{X} = \bigcup_{i=1}^{n} \mathsf{X}_i$
- tpc *is defined by* $\mathsf{tpc}[q_1, \ldots, q_n] = \bigwedge_{i=1}^{n} \mathsf{tpc}_i[q_i]$
- $\longrightarrow_\gamma$ *is defined by the rules:*

$$\frac{a = \{a_i\}_{i \in I} \in \gamma \qquad g = \bigwedge_{i \in I} g_i \qquad r = \bigcup_{i \in I} r_i \qquad \forall i \in I\ .\ q_i \xrightarrow{a_i, g_i, r_i}_i q_i' \qquad \forall i \notin I\ .\ q_i' = q_i}{(q_1, \ldots, q_n) \xrightarrow{a, g, r}_\gamma (q_1', \ldots, q_n')}$$

A composition $M = \gamma(M_1, \ldots, M_n)$ of abstract models M_i, $1 \leq i \leq n$ executes interactions $a = \{a_i\}_{i \in I} \in \gamma$ which corresponds to synchronizations of actions a_i of models M_i, $i \in I$. An interaction $a = \{a_i\}_{i \in I} \in \gamma$ is enabled from a state of M if all actions a_i are enabled.

In a composite model $M = \gamma(M_1, \ldots, M_n)$, many interactions can be enabled at the same time introducing a degree of non-determinism in the behavior of

M. In order to restrict non-determinism, we introduce priorities that specify which interaction should be executed among the enabled ones. A priority on $M = \gamma\ (M_1, \ldots, M_n)$ is a relation $\pi \subseteq \gamma \times \mathsf{Q} \times \gamma$ such that for all q the relation $\pi_q = \{\ (a, a') \mid (a, q, a') \in \pi\ \}$ is a partial order. We write $a\pi_q a'$ for $(a, q, a') \in \pi$ to express the fact that a has weaker priority than a' at state q. That is, if both a and a' are enabled at state q, only the action a' can be executed. Thus, priority $a\pi_q a'$ is applied only when the conjunction of the guards of a and a' is true. Let $q \xrightarrow{a,g,r}_\gamma q'$ and $q \xrightarrow{a',g',r'}_\gamma q''$ be transitions of M. Applying priority $a\pi_q a'$ boils down to transforming the guard g of a into the guard $g_\pi = g \wedge \neg g'$ and leaving the guard g' of a' unchanged.

Henceforth, we denote by $\mathsf{en}_q(a)$ the predicate characterizing the valuations of clocks for which an interaction a is enabled at state q. It is defined by:

$$\mathsf{en}_q(a) = \begin{cases} \mathsf{false} & \text{if } \nexists (q, a, g, r, q') \in \longrightarrow_\gamma \\ \bigvee_{(q,a,g,r,q') \in \longrightarrow_\gamma} g & \text{otherwise.} \end{cases}$$

Definition 4 (priority). *Given a composite model $M = (\gamma, \mathsf{Q}, \mathsf{X}, \longrightarrow_\gamma)$, the application of a priority π to M defines a new model $\pi M = (\gamma, \mathsf{Q}, \mathsf{X}, \longrightarrow_\pi)$ such that $\longrightarrow_\pi$ is defined by the rule:*

$$\frac{q \xrightarrow{a,g,r}_\gamma q' \qquad g_\pi = g \wedge \neg \bigvee_{a\pi_q a'} \mathsf{en}_q(a')}{q \xrightarrow{a,g_\pi,r}_\pi q'}$$

Example 2. Consider an abstract model $M = \pi\gamma(M_1, M_2, M_3)$ such that:

- abstract models M_1, M_2, and M_3 are provided by Figure 2,
- interactions $\gamma = \{a_1, a_2, a_3\}$ are defined by $a_1 = \{sync_1, sync_2, sync_3\}$, $a_2 = \{p, q\}$ and $a_3 = \{r, s\}$,
- priority π is such that $a_2\pi_q a_3$ for any control location q of M.

From the initial state $(q_1^1, q_2^1, q_3^1, 0)$, it can be easily shown that the execution sequences of M have the following form: $((q_1^1, q_2^1, q_3^1), 0) \xrightarrow{a_1} ((q_1^2, q_2^2, q_3^2), 0) \xrightarrow{5} ((q_1^2, q_2^2, q_3^2), 5) \xrightarrow{a_3} ((q_1^2, q_2^3, q_3^1), 5) \xrightarrow{\delta_2} ((q_1^2, q_2^3, q_3^1), 5 + \delta_2) \xrightarrow{a_2} ((q_1^1, q_2^1, q_3^1), 5 + \delta_2) \xrightarrow{a_1} ((q_1^2, q_2^2, q_3^2), 0)$, where $5 \leq \delta_2 \leq 15$. Notice that control location *err* cannot be reached in M_2 due to the application of priority $a_2\pi_q a_3$ for $q = (q_1^2, q_2^2, q_3^2)$.

3 The BIP Toolchain

This section presents the toolchain available with the BIP framework (see Figure 3). It consists in a rich set of tools for modeling, executing and verifying BIP models. The frontend of the toolchain is the parser which takes as input textual representations of BIP models according to the BIP grammar, and builds BIP models which are implemented using the EMF meta-modeling technology. Such models are the input for the rest of the tools, which fall into two main categories.

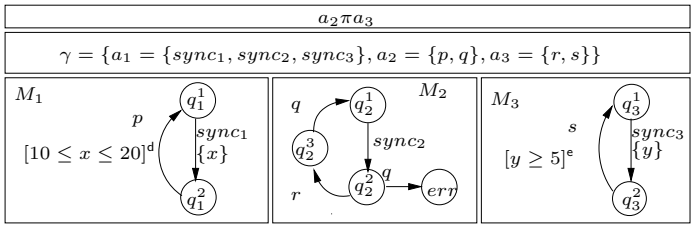


Fig. 2. Example of composition of abstract models with priorities

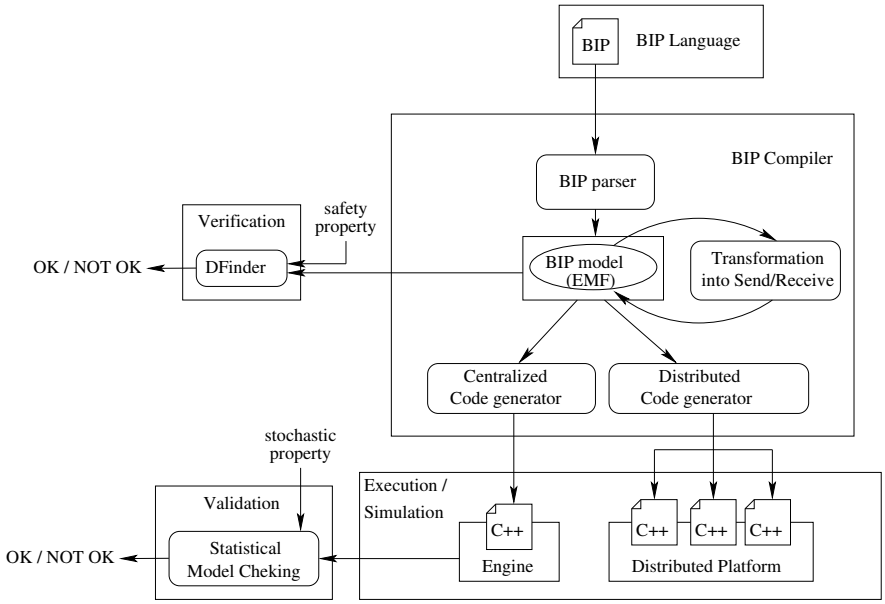


Fig. 3. Overview of the BIP Toolchain

Code generators. The BIP toolchain provides code generators for simulation and/or execution of models on target platforms. The standard code generator produces C++ code that relies on an engine for its execution. The centralized engine directly implements the operational semantics of BIP. It plays the role of the coordinator in selecting and executing synchronizations between the components, taking into account interactions and priorities specified in the input model. It supports both single-threaded and multi-threaded execution modes.

We have also developed a code generator for distributed platforms. It allows the transformation of BIP models into a set of standalone programs communicating through message passing which is implemented using the primitives available on the target platform. Such transformation has been proven correct, that is, it preserves the semantics of the input model.

Verification and validation tools. The BIP toolchain is completed by verification and validation tools for both checking system correctness and performance evaluation.

D-Finder is a verification tool targeting safety properties, e.g. deadlock freedom or mutual exclusion. The verification method implemented by D-Finder is based on the computation of invariants used to approximate the set of reachable states of the target system, hence the method is sound but not complete: it may not be able to prove a property even if it is satisfied by the system. Invariants are computed following the architecture of the system, that is, we generate invariants for components and for interactions. The approach is compositional and can be applied incrementally, allowing to better scale to large systems than traditional verification techniques.

In addition to D-Finder, the BIP toolchain includes the statistical model-checker SMC-BIP for checking stochastic properties expressed as probabilistic bounded linear temporal logic (PBLTL) formulas. Given a stochastic BIP model, a PBLTL formula and confidence parameters, SMC-BIP computes execution sequences until the formula can be proven with the target degree of confidence. Such a tool is particularly suited for evaluating quantitative properties including system performance related metrics.

4 Case Studies

In the following, we illustrate the use of our approach and tools through various robotics case studies. We used D-Finder to formally prove the correctness of a non trivial protocol between collaborating robots, as shown in Section 4.1. The statistical model-checker SMC-BIP was used to evaluate the performance and to fine-tune the strategy for the deployment of a swarm of robots (Section 4.2). In Section 4.3 we used our C++ code generator for deriving correct by construction distributed implementations from high-level models.

4.1 Compositional Verification of Safety Properties

We applied the compositional verification techniques for timed systems presented in [14] to a robotics scenario. It consists of cooperating robots used in a child's bedroom for home automation, automatic cleaning, or child assistance in tidying up. We considered the following types of robots/devices in the room, all capable of wireless communications.

Cleaning Robot. We assume the presence of an autonomous vacuum cleaner (e.g. Roomba) that can cooperate with other types of robots.

Toy Case Robot. The toy case robot—called Ranger—is currently developed in a research project of EPFL [15]. Its goal is to encourage the child to put away the toys in the case. We also assume that this robot has sensors able to detect the presence of the child when he is close enough.

Bed and desk chair. They are equipped with sensors allowing to detect when the child sits on.

Door. The bedroom door is equipped with an electric closing and locking system. A safety mechanism stops any closing procedure if the child tries to enter the bedroom while the door is closing.

Ceiling Camera. A camera located on the ceiling takes pictures at a given period P. They can be analyzed to detect whether the child is in the bedroom. The shape of the child can be tracked in these pictures only if it is not too close to other shapes (i.e. the toy case, on the bed and the chair).

In this scenario we were interested in a safety property stating that the child should not be in the bedroom while the cleaning robot is cleaning. To this end, we designed a protocol in which the cleaning robot (1) checks if child is outside the bedroom by correlating information from all the other robots / devices, (2) if so, closes and locks the door to keep the child outside, and (3) cleans the bedroom. We used formal verification to prove that our protocol satisfies the safety property.

The first thing one can observe in this system is that knowledge—e.g. the presence of the child—is distributed amongst the robots. One major issue for the cleaning robot is to build a consistent view of the status of the child (inside or outside) from local knowledge of the robots, and all this in real-time. We assume continuous sensing of the child for the case, the bed and the chair. On the other hand, pictures are taken only at specific time instants meaning that we have to deal with outdated information for the camera. If the child is not on a picture taken at a given time, then it was either outside, or inside and playing with the case, or on the bed or the chair. If we want to be sure that the child was outside the bedroom at the time the picture was taken, we need to know what was the status of the sensors of the case, the bed and the chair at this time. For this purpose we associate one timer to each sensor and reset it each time the child leaves. We also used a freshness parameter F for controlling the

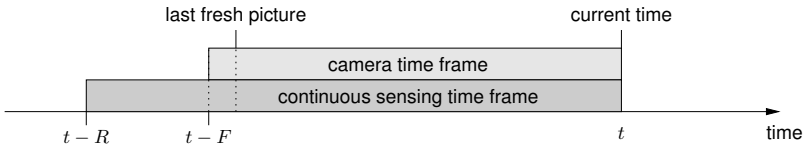


Fig. 4. Freshness parameters F for the camera and R for the other devices.

knowledge of the camera: the child is considered outside by the camera if he was not in a picture taken at most F time units ago. In a slightly different way, we used parameter R for the case, the bed and the chair: the child is considered outside by these devices if he was not detected for at least R time units. Notice that if $R \geq F$ we can safely conclude whether the child was outside or inside at the time the last picture was shot from the camera (see Figure 4). We also use R for the door, that is, it is considered closed if it was closed for more than R time units.

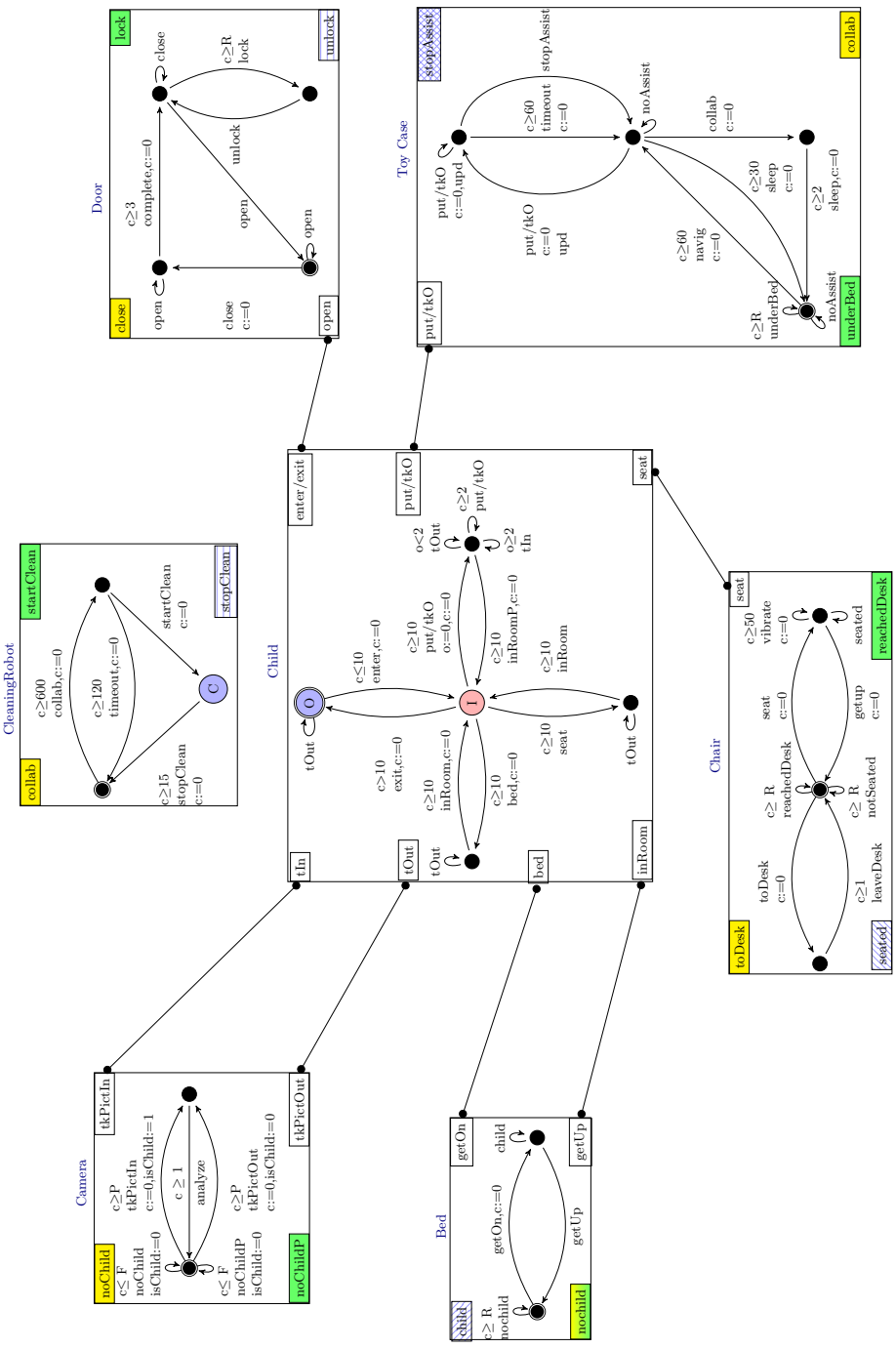


Fig. 5. BIP model of the cooperating robots example.

We built a BIP model for verifying the principles of the proposed protocol at high level (see Figure 5). If the child is not detected by all the devices (w.r.t. F and R), the cleaning robot starts locking the door since there is a high probability that the child is not in the bedroom at the current time (we are sure that at some instant in the last F time units, the child was not in the bedroom). This is represented by a strong synchronization between ports collab, close and noChild (the yellow ports of Figure 5). Notice that the behavior for parameters F and R is ensured by local conditions based on components clocks. Once executed, the door starts closing, and the case and the chair move towards locations that ease the cleaning robot to operate. Then the cleaning robot starts cleaning only if the child is still not detected by the devices and the door is still closed, considering again parameters F and R. If so, it locks the door and starts cleaning, which is modelled by a strong synchronization between ports startClean, lock, underBed, reachedDesk, noChild, noChildP (the green ports of Figure 5). Otherwise, if the cleaning is not possible for 120 time units, the cleaning robot timeouts and returns to its initial state. Intuitively this protocol is safe (provided $R \geq F$) since the cleaning starts only if the child was outside when the last picture was taken and the door was kept closed since this time. Moreover, during cleaning, the door remains closed by using the locking mechanism.

Using verification technique of [14] we managed to prove formally that the child is not in the bedroom while the robot is cleaning, provided $R \geq F$. More precisely, if the cleaning robot is in control state C, then the child is in state 0 (these control states are in blue in Figure 5). This property is non trivial as it strongly depends on the individual behavior of all the devices and in particular their timings, and it can be tricky to ensure for the system. We did several attempts before we obtained a correct design. For instance, we started with discreted and periodic sensing instead of continuous sensing. The flaw in this design was difficult to detect by simulation as it very rarely led to a violation of the safety property. Verification tools helped us in finding and fixing the problem.

Notice that the model proposed here is far too abstract to be used directly for implementing the devices. It uses primitives such as atomic synchronizations between two or more components (i.e. multi-party interactions) that should be translated into simpler interactions (e.g. messages passing). To get correct-by-construction implementations we could transform the proposed BIP model into a Send/Receive BIP model using techniques developed for generating distributed implementations from BIP, as presented explained in Section 4.3.

4.2 Quantitative Analysis of a Deployment Scenario

We considered a robotics scenario in which a swarm of marXbots [16] should (1) be deployed to find 5 victims (which are other marXbots) distributed all over an arena shown in Figure 6,

and (2) rescue the victims. In this scenario, we assume that the robots cannot use localization mechanisms (e.g. GPS, SLAM, etc.). Instead, during the deployment phase some of the robots stop and become landmarks, i.e., they are used to guide other robots for exploring the arena and rescuing the victims.

We used the statistical model-checking tool SMC-BIP to analyze to the deployment phase only. We first built a BIP model of a single marXbot including a faithful implementation of its sensors. Following the approach implemented in the simulator ARGoS [17], we rely on synchronous discrete time execution with a duration of 10 ms for the time steps. The model of the swarm represents 1500 lines of BIP code along with 1200 lines of external C++ code.

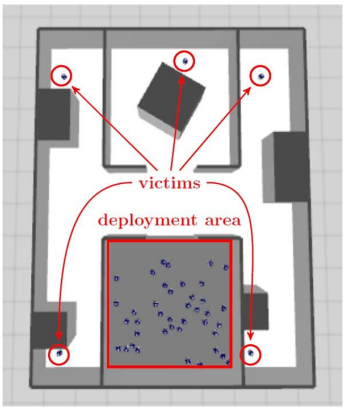
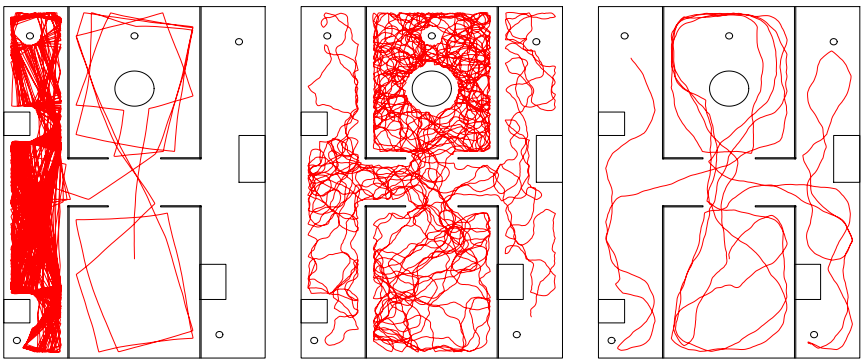


Fig. 6. Arena of the scenario.

Single robot behavior. We started by experimenting with several behavioral strategies for a single robot: straight walk, random walk, and random walk improved using the rotating scanner. All includes basic obstacle avoidance so as not to bump into walls and/or other robots. Figure 7 shows examples of simulations obtained for the different strategies, where the path followed by the explorer is represented by the red drawing and the victims are represented by the five small black circles (three at the top and two at bottom of the arena). Using straight walk minimize the distance for travelling from one location to another. However, it resulted in a very poor coverage since the explorer was trapped on the left side of the arena from which it did not escape even after a long time. Random walk led to good coverage but longer delays for finding the first three victims than the ones obtained with straight walk. We improved random walk by using the rotating scanner which allows the explorer to track long distances obstacles and to follow corridors and walls, which is clearly visible on simulations (see Figures 7). All these observations are confirmed by the analysis performed by SMC-BIP with which we computed the expected time for finding the 1^{st} and the 5^{th} considering probability 0.85, provided in Figure 8. Parameters α, β and δ in table of Figure 8 correspond to the target degree of confidence for SMC-BIP. The lower these parameters are, the lower the probability to obtain an incorrect answer is. They are formally defined in [18]. Using SMC-BIP we also managed to show that increasing the number of explorers (we tested for 11, 21, and then 31) tends to reduce the expected delays for finding victims (see Figure 8).

Cooperation between robots. We completed the model by including landmarks behavior and corresponding communications. If a robot become too far away from other landmarks, or if it finds a victim, it stops to establish a new landmark. The goal of these landmarks during the deployment phase is to avoid exploring areas that have been already explored. Landmarking alone reduced drastically the performances, as shown by Figure 8. This can be explained by the fact that landmarking reduces the moving range of the explorers and decreases the number of active robots, sometimes to the point where all robots were stopped (i.e. were

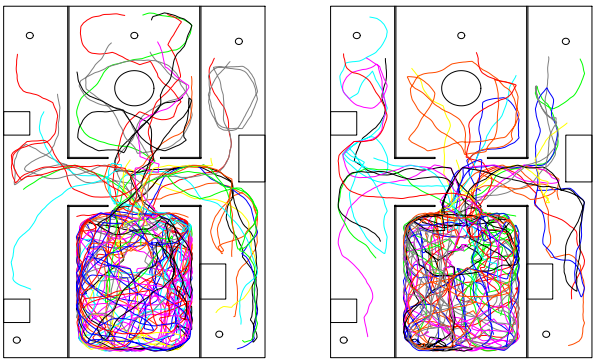


(a) straight (b) random (c) random + scanner

Fig. 7. Simulation of a single robot and various moving strategies.

strategy:	straight	random	random + scanner				landmark	comm.
number of explorers:	1	1	1	11	21	31	31	31
1^{st} victim (α=β=δ=0.05)	343	2996	892	211	188	152	?	375
5^{th} victim (α=β=δ=0.01)	timeout	41250	11562	1171	820	742	timeout	1797

Fig. 8. Delays in seconds for finding victims with probability P=0.85.



(a) landmark (b) landmark+comm.

Fig. 9. Simulation of landmarking strategies for 31 explorers.

landmarks) whereas victims remained to be found. An example of such situation can be observed in Figure 9(a).

The goal of landmarking is mainly to reduce the time to accomplish the second phase of the scenario. To this purpose, landmarks must communicate with active robots to route them for achieving their goal (exploring, rescuing, etc.). We included basic communication capabilities in the model allowing landmarks to route back robots if there is no need for exploration in their given direction (e.g. presence of a dead end). These communications were implemented by simple binary connectors between the robots. Adding communications allowed acceptable performances for finding all the five victims, while establishing landmarks required by the second phase of the scenario. Simulation traces clearly show the switchbacks performed by the robots when meeting landmarks from which no further exploration is needed (see Figure 9(b)).

SMC-BIP allowed us to fine-tune the behavior of the marXbot to optimize the deployment phase of the scenario. Such fine-tuning is also possible with standard simulation techniques (e.g. with ARGoS), but statistical model-checking permits us to have reliable information about the performances of the swarm, guaranteed by explicit degrees of confidence and based on exploration of possible behaviors. For example, it required sometimes more than 20000 simulations for SMC-BIP to conclude on a single delay value. The BIP model we developed can also be a basis for computing stochastic abstractions and/or for applying verification techniques and tools.

4.3 Collaborating Robots

Our third case study is a robotic application that consists of a set of communicating robots that collaborate to perform a given task. The scenario is described as following: initially, the robots, with blue color, are dispatched in an arena with different positions. They start by exploring the arena in order to find each others. When 3 robots become sufficiently close, they group themselves to form a "V" form and change their color to red. Then, they go towards an object (e.g a ball) which is positioned at the center of the arena, and push it. Finally, the other robots go towards the border of the arena when they "see" the red robots. We assume that the robots are equipped by proximity sensors to detect obstacles (arena's walls and the ball), a camera to detect the robot's colors and a led to change the color.

Our case study is composed of 6 instances of robot. Figure 10 shows the BIP model of a single robot. We used timing constraints and time progress conditions to express timeout when the grouping action cannot be performed within a given amount of time.

The grouping of robots is modeled by a connector that synchronizes group transitions of any 3 robots. The connector enables the interaction between the robots only if they are sufficiently close to each others. As shown in Figure 11 the connector is guarded by a guard on robot's positions that determines if the robots are close to each others.

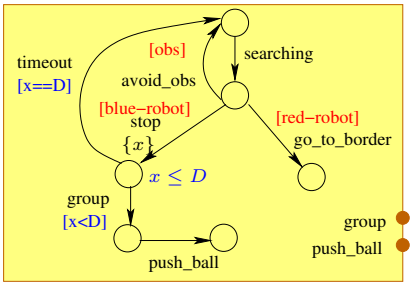


Fig. 10. Model of a single robot.

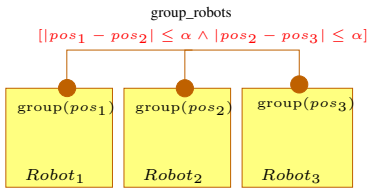


Fig. 11. Synchronization of robots for achieving their grouping.

The challenging issue in this application is to come up with distributed implementation that correcly achieves the expected synchronization of three robots. Following [13], instead of writing directly the distributed implementation, we used BIP connectors to express the grouping of three robots on high-level models, and were able to generate all the communications needed for its realization at runtime.

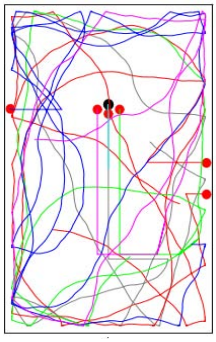


Fig. 12. Experimental results for 6 robots.

In order to make simulations as realistic as possible, we also modeled robot's behavior, such as robot's movement, sensor's reading and camera image processing. Figure 12 presents the simulation results for 6 robots, where the red circles represent the final positions of the robots and the black one represents the ball. It shows that the robots effectively managed to group themselves and push the ball.

5 Conclusion

We have presented a rigorous system design flow for timed systems. It is based on the BIP language in which the notion of behavior—expressed a set of components—is clearly separated from the notion of architecture—expressed by interactions and priorities. Correct implementations are obtained by *(i)* checking the design on abstract models using verification tool D-Finder, and *(ii)* refining such models using proven correct source-to-source transformers. In addition, system performances can also be evaluated at design time using statistical model-checker SMC-BIP. In this paper, we showed how this framework was successfully applied to robotics case studies.

As future work, we plan to improve our method for the generation of distributed implementations by considering non perfectly synchronized clocks and disconnected communication networks.

References

1. Halbwachs, N.: Synchronous Programming of Reactive Systems. Kluwer Academic Publishers (1993)

2. Burns, A., Welling, A.: Real-Time Systems and Programming Languages. Addison-Wesley (2001) 3rd edition.
3. Bliudze, S., Sifakis, J.: A Notion of Glue Expressiveness for Component-Based Systems. In: CONCUR'08. Volume 5201 of LNCS., Springer (2008) 508–522
4. Garlan, D., Monroe, R., Wile, D.: ACME : An architecture description interchange language. In: CASCON'97. (1997) 169–183 see also http://www.cs.cmu.edu/ acme/.
5. Magee, J., Kramer, J.: Dynamic structure in software architectures. In: SIGSOFT'96. (1996) 3–14
6. OMG: OMG Systems Modeling Language SysML (OMG SysML). Object Management Group (2008)
7. Feiler, P.H., Lewis, B., Vestal, S.: The SAE Architecture Analysis and Design Language (AADL) Standard: A basis for model-based architecture-driven embedded systems engineering. In: RTAS Workshop on Model-driven Embedded Systems. (2003) 1–10 see also http://www.sae.org.
8. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity: The Ptolemy approach. Proceedings of the IEEE **91**(1) (2003) 127–144
9. Basu, A., Gallien, M., Lesire, C., Nguyen, T.H., Bensalem, S., Ingrand, F., Sifakis, J.: Incremental Component-Based Construction and Verification of a Robotic System. In: ECAI'08. Volume 178 of FAIA., IOS Press (2008) 631–635
10. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the bip framework. IEEE Software **28**(3) (2011) 41–48
11. Bensalem, S., de Silva, L., Griesmayer, A., Ingrand, F., Legay, A., Yan, R.: A formal approach for incremental construction with an application to autonomous robotic systems. In Apel, S., Jackson, E.K., eds.: Software Composition. Volume 6708 of Lecture Notes in Computer Science., Springer (2011) 116–132
12. 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
13. Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: From high-level component-based models to distributed implementations. In: EMSOFT. (2010)
14. Astefanoaei, L., Rayana, S.B., Bensalem, S., Bozga, M., Combaz, J.: Compositional invariant generation for timed systems. In: TACAS. (2014) to appear.
15. : Intelligent robots for improving the quality of life. `http://www.nccr-robotics.ch`
16. Bonani, M., Longchamp, V., Magnenat, S., Rtornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., Mondada, F.: The MarXbot, a Miniature Mobile Robot Opening new Perspectives for the Collective-robotic Research. In: International Conference on Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ. IEEE International Conference on Intelligent Robots and Systems, IEEE Press (2010) 4187–4193
17. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Caro, G.D., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: Argos: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence **6**(4) (2012) 271–295
18. Bensalem, S., Bozga, M., Delahaye, B., Jégourel, C., Legay, A., Nouri, A.: Statistical model checking qos properties of systems with sbip. In Margaria, T., Steffen, B., eds.: ISoLA (1). Volume 7609 of Lecture Notes in Computer Science., Springer (2012) 327–341