Technical Report
Architecture-based Rigorous System Design: An Airplane Engine Control Software Case Study

No Author Given

No Institute Given

Abstract. In this case study, we apply architecture-based rigorous system design approach to the control software of the airplane engine controller. Rigorous system design refers to a formal model-based process that leads from requirements to correct implementations. Architectures are a means for ensuring global coordination properties of system models and thus, achieving design correctness by construction. We implement the architecture diagrams for airplane engine control software in the Behavior-Interaction-Priority (BIP) component-based framework. We show how to obtain a system model by applying architectures to a set of atomic components. We also perform deadlock freedom analysis of the resulting system model by using the BIP tool-set and additionally validation analysis of our approach by using nuXmv model checker to verify that the safety properties enforced by the architecture are indeed satisfied.

1 Introduction

Design, manufacture and verification of large scale reliable hardware/software systems (e.g. cyber-physical systems) remains a grand challenge in system design automation [10]. To address this challenge, the rigorous system design methodology [9]. Rigorous system design can be understood as a formal, accountable and coherent process for deriving correct and trustworthy system implementations from high-level specifications. The essential safety properties of the design are guaranteed at the earliest possible design phase by applying algorithmic verification to the system model, and then the system implementation is automatically generated by a sequence of property preserving model transformations, progressively refining the model with details specific to the target platforms.

The architecture-based design approach consists of the three stages. First, architecture styles relevant for the application domain are identified and formally modelled. Ideally, this stage is only realised once for each application domain. The remaining stages are applied for each system to be designed. In the second design stage, requirements to be satisfied by the system are analysed and formalised, atomic components realising the basic functionality of the system are designed (components previously designed for other systems can be reused) and used as operands for the application of architectures instantiated from the styles defined in the first stage. The choice of the architectures to apply is driven by the requirements identified in the second stage. Finally, the resulting system is checked for deadlock-freedom. Properties, which are not enforced by

construction through architecture application, must be verified a posteriori. In this case study, we illustrate all steps of this process, except the requirement formalisation.

The Behavior-Interaction-Priority (BIP) framework [2] is proposed as a component-based system design framework, which comes with a formal language with well-defined semantics and a tool chain supporting rigorous system design process. The BIP language offers a three-layered modeling mechanism for constructing complex system behavior and architectures [6], i.e., Behavior, Interaction, and Priority. Behavior is characterized by a set of components, which are formally defined as automata extended with linear arithmetic. Interaction specifies the multiparty synchronization of components, among which data transfer may take place. Priority can be used to schedule the interactions or resolve conflicts when several interactions are enabled simultaneously. The key insight underlying this three-layered modeling mechanism is the principle of separation of concerns, that is, system computation is captured by a set of components, and system coordination is modeled by interaction and priority.

BIP framework advocates component-based design of correct-by-construction applications. It provides a simple, but powerful mechanism for the coordination of concurrent components by superposing three layers. First, component behaviour is described by Labelled Transition Systems (LTS) having transitions labelled with ports. Ports form the interface of a component and are used to define its interactions with other components. Second, interaction models, i.e. sets of interactions, define the component coordination. Interactions are sets of ports that define allowed synchronizations between components. An interaction model is defined in a structured manner by using connectors. Third, priorities are used to impose scheduling constraints and to resolve conflicts when multiple interactions are enabled simultaneously.

The BIP tool chain supports both algorithmic verification and testing of high-level system designs [3, 4, 8] and automatic system synthesis of low-level implementations from high-level system designs [5]. In practice, BIP has been actively used in several applications [1, 7].

The rest of the paper is structured as follows. In Section **??** and Section **??**, we introduce the preliminaries and the BIP modeling language. In Section **??** and Section **??**, we review the most related works and draw some conclusions and outline directions for future work.

2 Architectures in BIP

We use architecture diagrams [] to model the architecture styles in BIP. An architecture diagram consists of a set of component types, with associated cardinality constraints representing the expected number of instances of each component type and a set of connector motifs. Connector motifs, which define sets of BIP connectors, are non-empty sets of port types, each labelled as either a trigger or a synchron. Each port type has a cardinality constraint representing the expected number of port instances per component instance and two additional constraints: multiplicity and degree, represented as a pair m : d. Multiplicity constrains the number of instances of the port type that must participate in a connector defined by the motif; degree constrains the number of connectors attached to any instance of the port type.

In this section, we present the BIP model with multiparty synchronization and data transfer. A BIP model is a parallel composition of a set of components. A BIP component is formally defined as an automaton extended with linear integer arithmetic as follows.

Definition 1 (BIP component). *Given a finite set of variables $\mathbb{V}$, a BIP component is defined as a tuple $B = \langle \mathbb{V}, \mathbb{L}, \mathbb{P}, \mathbb{E}, \ell \rangle$, where 1. $\mathbb{L}$ is a finite set of control locations; 2. $\mathbb{P}$ is a finite set of communication ports; 3. $\mathbb{E} \subseteq \mathbb{L} \times \mathbb{P} \times \mathcal{F}_{\mathbb{V}} \times \mathcal{E}_{\mathbb{V}} \times \mathbb{L}$ is a finite set of transition edges extended with guards in $\mathcal{F}_{\mathbb{V}}$ and operations in $\mathcal{E}_{\mathbb{V}}$; 4. $\ell \in \mathbb{L}$ is an initial control location.*

Transition edges in a component are labeled by ports, which form the interface of the component. We assume that, from each control location, every pair of outgoing transitions have different ports, and the ports of different components are disjoint. In other words, transitions with the same ports in the component are not enabled simultaneously. Given a component violating such assumptions, one can easily transform it into the required form by renaming the ports, while retaining the BIP expressiveness power. To ease the presentation, we denote in the sequel the id of the unique component where port p is defined by $id(p)$.

We denote by $\mathcal{B} = \{B_i \mid i \in [1, n]\}$ a set of components. In BIP, coordinations of components are specified by using interactions.

Definition 2 (Interaction). *An interaction for $\mathcal{B}$ is a tuple $\gamma = \langle g, \mathcal{P}, f \rangle$, where $g \in \mathcal{F}_{\mathbb{V}}$, $f \in \mathcal{E}_{\mathbb{V}}$ and $\mathcal{P} \subseteq \bigcup_{i=1}^{n} \mathbb{P}_i$, $\mathcal{P} \neq \emptyset$, and for all $i \in [1, n]$, $|\mathcal{P} \cap \mathbb{P}_i| \leq 1$.*

Intuitively, an interaction defines a guarded multiparty synchronization with data transfer: when the guard g of an interaction $\mathcal{P}$ is enabled, then the data transfer specified by f can be executed, and after that the transitions labelled by the ports in γ can be taken simultaneously. We denote by Γ a finite set of interactions. A BIP model is constructed by composing a number of components with interactions.

Definition 3 (BIP Model). *A* BIP *model $\mathcal{M}_{\text{BIP}}$ is a tuple $\langle \mathcal{B}, \Gamma \rangle$, where $\mathcal{B}$ is a finite set of components, and Γ is a finite set of interactions for $\mathcal{B}$.*

We do not take priority into account in this paper, as in the previous work [3, 8], since adding priority will not introduce any errors. If a model without priority is safe, then after adding priority constraints it remains safe. We use a simple mutual exclusion protocol to illustrate BIP.

A state of a BIP model is a tuple $c = \langle \langle l_1, \mathbf{V}_1 \rangle, \ldots, \langle l_n, \mathbf{V}_n \rangle \rangle$, where for all $i \in [1, n]$, $l_i \in \mathbb{L}_i$ and $\mathbf{V}_i$ is a valuation of $\mathbb{V}_i$. A state c_0 is initial if for all $i \in [1, n]$, $l_i = \ell_i$ and $\mathbf{V}_i$ is the initial valuation of $\mathbb{V}_i$. A state c is an error if for some $i \in [1, n]$, l_i is an error location. We say an interaction $\gamma \in \Gamma$ is enabled on a state c if for every component $B_i \in \mathcal{B}$, such that $\gamma \cap \mathbb{P}_i \neq \emptyset$, there is an edge $\langle l_i, \gamma \cap \mathbb{P}_i, g_i, f_i, l_i' \rangle \in \mathbb{E}_i$ and $\mathbf{V}_i \models g_i$. The labeled transition system semantics of a BIP model is defined as follows.

Definition 4 (BIP operational semantics). *Given a* BIP *model $\mathcal{M}_{\text{BIP}} = \langle \mathcal{B}, \Gamma \rangle$, its operational semantics is defined by a labeled transition system $\mathcal{T}_{\text{BIP}} = \langle \mathcal{C}, \Sigma, \mathcal{R}, \mathcal{C}_0 \rangle$, where*

1. *$\mathcal{C}$ is the set of states,*
2. *$\Sigma = \Gamma$,*
3. *$\mathcal{R}$ is the set of transitions, and we say that there is a transition from a state c to another state c', if there is an interaction γ such that,*
   (a) *γ is enabled in c;*
   (b) *for all $B_i \in \mathbb{B}$ such that $\gamma \cap \mathbb{P}_i \neq \emptyset$, there is an edge $\langle l_i, \gamma \cap \mathbb{P}_i, g_i, f_i, l_i' \rangle \in \mathbb{E}_i$, then $\mathbf{V}_i' = \mathbf{V}_i[\mathbb{V}/f_i(\mathbb{V})]$;*
   (c) *for all $B_i \in \mathbb{B}$ such that $\gamma \cap \mathbb{P}_i = \emptyset$, $l_i' = l_i$ and $\mathbf{V}_i' = \mathbf{V}_i$.*
4. *$\mathcal{C}_0$ is the set of initial states.*

In this paper, we do not use temporal logics to specify safety properties, but recognize a set of locations as error locations. A BIP model is safe if no error states are reachable. Notice that any safety property can be encoded as a reachability problem by necessarily creating additional components.

3 Case study

In this section, we introduce an airplane engine control software case study. For presentation simplicity, we choose one of its subsystems, the PLA signal processing system, as representative. We will give an informal description, focusing on its functional structure, properties and conditional constraints. Then we give the formal design model in BIP. Finally, we provide model verfication and validation.

3.1 Informal description

The Power Level Angle (PLA) signal processing system is part of the Full Authority Digital Engine Control System (FADEC), which provides engine control during flight to achieve stable and transient engine characteristics. The purpose of PLA signal processing system is to convert the obtained throttle stick signal value into the original throttle stick angle value and judge it. If it is judged that there is no fault, then it will return the corresponding value of the PLA signal after processing and set a fault signal to invalid, representing that the system has no PLA signal fault in current period. Otherwise it will return the fault information and set the fault signal to valid.

There are three modules in the system. PLA signals will first undergo a BIT Diagnosis module, then be calibrated and converted into physical quantities. After the Extremum and Slope Diagnosis module, it will finally generate the PLA fault signal for output.

The Built-int Test(BIT) refers to the detection and monitoring of the system and its own equipment. In the PLA signal processing system model, the BIT Diagnosis realizes the analog quantity, or periodic fault detection of circuits such as fault location and processing based on the diagnosis results.

The calibration conversion module's function is to collect the signal from signal collector. The arriving signal will be converted to the corresponding engineering value by means of a calibration curve or an index table.

An Extremum/Slope Diagnosis module contains extremum diagnosis and slope diagnosis.The function of extremum diagnosis is to judge whether the current signal is

in valid range since the PLA signal processing system cannot handle illegal data. The function of slope diagnosis is to give the constraint of the magnitude of the change between the two adjacent signals.

The three modules introduced above work together for giving a credible output signal. In the next chapter, we will translate this informal model to formal BIP model for further verification and validation.

3.2 Formal system model in BIP

We construct the BIP model of the PLA signal processing system through the composition of atomic and composite components with BIP connectors. The block diagram for our system is shown in Fig.1, it consists of a Task Unit and a Signal Processing Unit. The Signal Processing Unit consists of atomic component c1, composite components p1 and c2. In the Task Unit, there is an atomic component t1, which produce signal value for Signal Processing Unit.

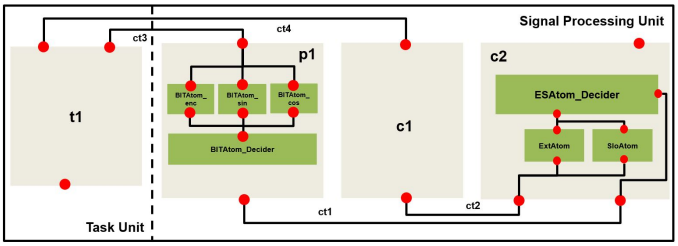

Fig. 1. System Architecture

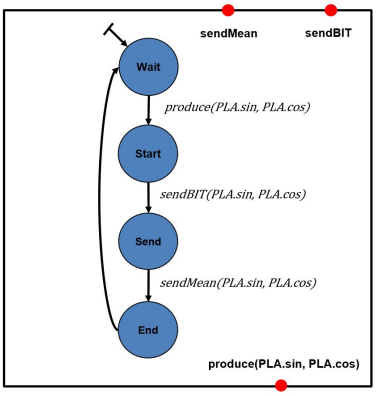

Fig. 2. The Task Component

The behavior of atomic component t1 is shown in Fig.2. It is used as simulator of the input stimuli. In the graphical notation, the atomic is initialized to *Wait*, and the following three transitions are driven by corrensponding events: *produce*, *sendBIT* and *sendMean*, which represents three corrensponding ports in the atomic. For instance, whenever the current state is at *Start*, the port *sendBIT* will be activated and the transition will be executed once the event *sendBIT* is done.

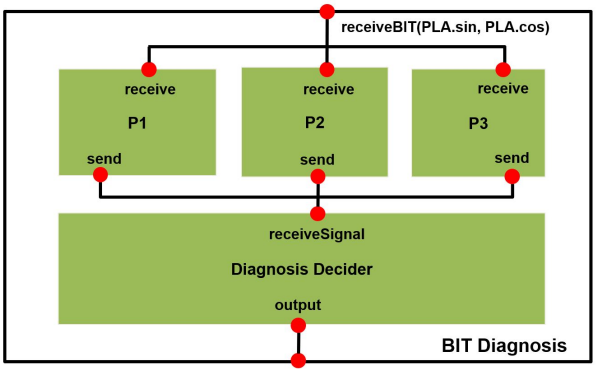

Fig. 3. Architecture of BIT Diagnosis

3.2.1 BIT Diagnosis module

In our design, the BIT Diagnosis module, Calibration Conversion module and Extremum/Slope Diagnosis module are three subsystems of PLA signal processing system. The BIT Diagnosis module is shown in figure 3. In the graphical notation, the three atomic component instances *p1*, *p2* and *p3* receive BIT signal through port *receiveBIT*, a connector distributes the signal to these components and another connector merges their output and sends it to the an instance of *Decider* component. The BIP description of the definition and instantiation of *BIT Diagnosis* is shown below:

```
compound BITCompound_P1
  component BITAtom_enc enc
  component BITAtom_sin sin
  component BITAtom_cos cos
  component BITAtom_Decider d

  connector ThreeToOne o(enc.sendBIT,sin.sendBIT,cos.sendBIT,d.receiveSignal)
  connector Merge m(enc.receiveBIT,sin.receiveBIT,cos.receiveBIT)

  export port m.Merged_Signal as receiveBIT
```

```
  export port d.output as sendBIT
end
```

We define the two ports in *BITCompound_P1* as export to give them an external property which enables them to be triggered by other components.

```
connector type ThreeToOne(PLAPort_t1 r1, PLAPort_t1 r2, PLAPort_t1
r3, PLAPort_t3 r4)
  define r1 r2 r3 r4
  on r1 r2 r3 r4 down {r4.a=r1.a; r4.b=r2.a; r4.c=r3.a;}
end


connector type Merge(PLAPort_t2 r1, PLAPort_t2 r2, PLAPort_t2
r3)
  data int mysin,mycos
  export port PLAPort_t2 Merged_Signal(mysin,mycos)
  define r1 r2 r3
  on r1 r2 r3 down {r1.a=mysin; r2.a=mysin; r3.a=mysin; r1.b=mycos;
r2.b=mycos; r3.b=mycos;}
end
```

In our system, atomic component is considered as the smallest unit of the architecture, while in the combination of several components, such as our BIT Diagnosis module, we have to ensure that the output of three components can simultaneously reach the connoector, otherwise it will fall into deadlock status. To avoid this problem, the three atomic components are designed homogeneous, we take one of these three components, *BITAtom_enc p1*, as an example, its ports and behavior is shown below:

Table 1. Transitions of P1

Transition	Guard	Action
1	PLA.sin $<$ 500$\wedge$ PLA.cos $<$ 500	$p := p + 1; q := 0$
2	PLA.sin $>=$ 500$\wedge$ PLA.cos $>=$ 500	$p := 0; q := q + 1$
3	PLA.sin $>=$ 500$\oplus$ PLA.cos $>=$ 500	$p := 0; q := 0$
4	$p >= 5$	PLA.Signal_enc: $= 1$
5	$q >= 5$	PLA.Signal_enc: $= 0$
6	$p < 5$	null
7	$q < 5$	null

During execution, we consider a WAIT to WAIT transition set as a cycle. In each cycle, we ensure that an *sendBIT* event will be activated to avoid the system falling into deadlock status. With the combination of connector, *P1* gives a synchronous output together with other two atomic component instances. We define two variables p and q as

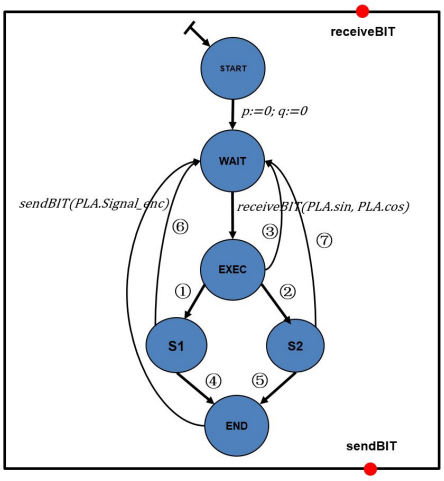

Fig. 4. BIT Diagnosis P1

counters to control the vary of signal. Their constraint indicates that for every continuous five cycles, the system does a setting and gives a corresponding output through port *sendBIT*.

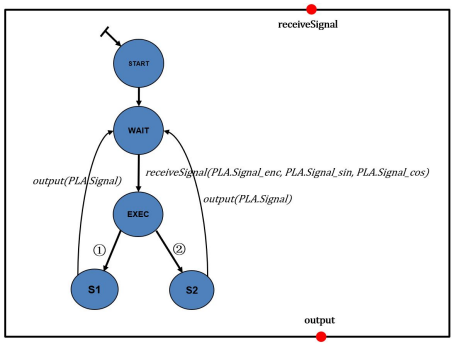

Fig. 5. BIT Diagnosis Decider

As shown in Fig.5, the BIT Diagnosis Decider is responsible for integrating the signals output by three BIT Diagnosis. We also produce a WAIT to WAIT transition set, *receiveSignal* as input event and *output* as output event. We define a bool type variable *PLA.Signal* as the BIT fault signal for output.

3.2.2 Calibration Conversion module

Table 2. Transitions of Decider

Transition	Guard	Action
1	PLA.Signal_enc $= 0 \wedge$ PLA.Signal_sin $= 0 \wedge$ PLA.Signal_cos $= 0$	$PLA.Signal := 0$
2	PLA.Signal_enc $= 1 \vee$ PLA.Signal_sin $= 1 \vee$ PLA.Signal_cos $= 1$	$PLA.Signal := 1$

The Calibration Conversion module collects the signal from signal collector and converts it to the corresponding engineering value by means of a calibration curve and an index table. This module is designed as an atomic component c1 in the BIP model, its ports and behavior is shown below:

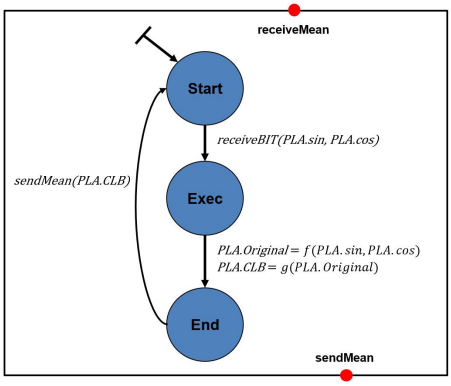

Fig. 6. Calibration conversion module

It has been introduced that every atomic component can be converted to its corresponding BIP description. The BIP description of the definition of *Calibration Conversion* is shown below:

```
atom CalAtom
  data int mysin,mycos
  data int myangle

  export port PLAPort_t2 receiveMean(mysin,mycos)
  export port PLAPort_t4 sendMean(myangle)

  place START, EXEC, END

  initial to START
    do {myangle = 0;}
```

```
  on receiveMean from START to EXEC

  internal from START to EXEC
    do {myangle = mysin * mycos + 1;}

  on sendMean from END to START
end
```

In the BIP description, we define internal data, external ports, serveral states and transitions. Transitions begin with field *on* are external transitions which are driven by external events. Relatively, transitions begin with field *internal* literally internal transitions which are executed as soon as the atomic component reach the state. We define an internal variable *myangle* for output. System calculate its value through calibration curve and index table from two variables *mysin* and *mycos*.

3.2.3 Extremum/Slope Diagnosis module

The Extremum/Slope Diagnosis module is almost the same as BIT Diagnosis module. It picks up signal value, diagnoses them and integrates the results into one variable for output. We also provide the architecture of Extremum/Slope Diagnosis:

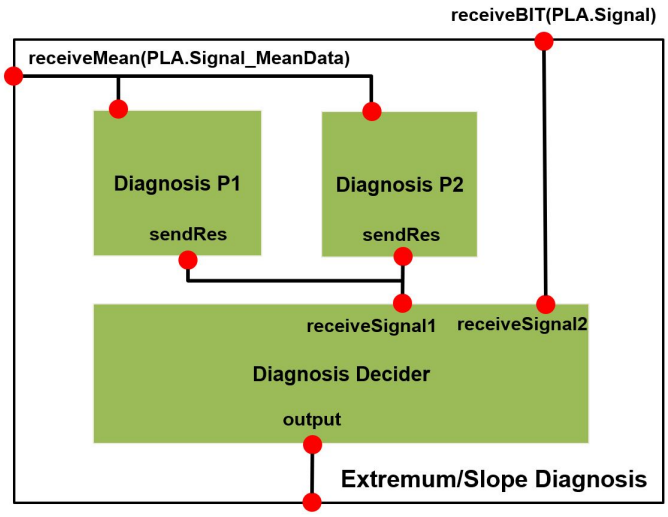

Fig. 7. Architecture of Extremum/Slope Diagnosis

It should be noticed that in Extremum/Slope Diagnosis module, besides signals sent by internal atomic components p1 and p2, the *Decider* also pick up signal from BIT Diagnosis module as input. We give the BIP Description of the Extremum/Slope Diagnosis module below:

```
compound ESCompound_C1
  component ExtAtom enc
  component SloAtom slo
  component ESAtom_Decider d

  connector TwoToOne o(ext.sendRes,slo.sendRes,d.receiveSignal1)
  connector Merge_two m(ext.receiveMean,slo.receiveMean)

  export port d.receiveSignal2 as receiveBIT
  export port m.Merged_Signal_two as receiveMean
end
```

The connector types *TwoToOne* and *Merge_two* do almost the same work as the two connector types used in BIT Diagnosis module except that the connector types used here have only two parameters, not three.

In the entire system, the compound type *ESCompound* will be instantiated as c2, which is introduced in Fig.1.

Till now, we have introduced our system informally and in formal BIP description. To give a formal proof whether our system and BIP model is safety and effectiveness or not, the verification and validation of the system above will produced in the next chapter.

3.3 Model verification and validation

4 Related work

In the BIP framework, DFinder [3] is a dedicated tool for invariant generation and deadlock-freedom detection. DFinder computes the system invariant in a compositional manner: it first computes a component invariant over-approximating the reachable states of each component and then computes an interaction invariant over-approximating the global reachable states. The system invariant is then the conjunction of all component invariants and the interaction invariant. Though being scalable for large system models, DFinder does not handle system models with data transfer, which hampers the practical application of DFinder and of the BIP framework, since data transfer is necessary and common in the design of real-life systems (e.g. message passing). Besides, it is not clear in DFinder how to refine the abstraction automatically when the inferred invariant fails to justify the property.

A compositional encoding of BIP into nuXmv and an efficient instantiation of Explicit Scheduler Symbolic Thread (ESST) framework for BIP have been presented in [4]. The encoding into nuXmv allows one to exploit the state-of-the-art verification techniques to verify BIP models. The ESST based technique encodes the components as preemptive threads with predefined primitive functions and utilizes a dedicated stateful BIP scheduler to orchestrate the abstract reachability analysis of the components. The scheduler interacts with components via primitive functions, and also respects BIP

operational semantics. Moreover, partial order reduction techniques are applied in the scheduler to reduce the states of scheduler.

In [8], we present a lazy predicate abstraction technique for BIP, and we also propose a novel reduction technique called simultaneous set reduction, which is further combined with lazy abstraction to reduce the search space of the abstract reachability analysis.

5 Conclusion and future work

References

1. Abdellatif, T., Bensalem, S., Combaz, J., Silva, L.d., Ingrand, F.: Rigorous design of robot software: A formal component-based approach. Robotics and Autonomous Systems 60(12), 1563–1578 (2012)
2. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the BIP framework. Software, IEEE (2011)
3. Bensalem, S., Bozga, M., Nguyen, T., Sifakis, J.: Compositional verification for component-based systems and application. IET Software (2010)
4. Bliudze, S., Cimatti, A., Jaber, M., Mover, S., Roveri, M., Saab, W., Wang, Q.: Formal verification of infinite-state BIP models. In: ATVA (2015)
5. Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: From high-level component-based models to distributed implementations. In: Proceedings of the 10th International conference on Embedded software, EMSOFT 2010, Scottsdale, Arizona, USA, October 24-29, 2010. pp. 209–218. ACM (2010)
6. Konnov, I., Kotek, T., Wang, Q., Veith, H., Bliudze, S., Sifakis, J.: Parameterized systems in bip: design and model checking. In: Proceedings of the 27th International Conference on Concurrency Theory (CONCUR 2016). pp. 30–1. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2016)
7. Lekidis, A., Stachtiari, E., Katsaros, P., Bozga, M., Georgiadis, C.K.: Model-based design of iot systems with the BIP component framework. Softw., Pract. Exper. 48(6), 1167–1194 (2018)
8. Qiang, W., Bliudze, S.: Verification of component-based systems via predicate abstraction and simultaneous set reduction. In: Trustworthy Global Computing, pp. 147–162. Springer (2015)
9. Sifakis, J.: Rigorous system design. Foundations and Trends in Electronic Design Automation (2013)
10. Sifakis, J.: System design automation: Challenges and limitations. Proceedings of the IEEE 103(11), 2093–2103 (2015)