Relating BIP and Reo

Kasper Dokter, Sung-Shik Jongmans, Farhad Arbab   Simon Bliudze

Centrum Wiskunde & Informatica,   École Polytechnique Fédérale de Lausanne,
Amsterdam, Netherlands   Lausanne, Switzerland

Coordination languages simplify design and development of concurrent systems. Particularly, exogenous coordination languages, like BIP and Reo, enable system designers to express the interactions among components in a system explicitly. In this paper we establish a formal relation between BI(P) (i.e., BIP without the priority layer) and Reo, by defining transformations between their semantic models. We show that these transformations preserve all properties expressible in a common semantics. This formal relation comprises the basis for a solid comparison and consolidation of the fundamental coordination concepts behind these two languages. Moreover, this basis offers translations that enable users of either language to benefit from the toolchains of the other.

1 Introduction

Context.   Over the past decades, architecture description languages (ADL) and coordination languages have emerged as fundamental tools for tackling complexity in the design of correct-by-construction componentised software systems [15]. However, no language has yet emerged as a de facto standard, and no consensus exists on how to properly design such languages, either. BIP [9, 10] and Reo [3] each addresses this complexity and provides a formal semantic framework, which allows reasoning about and proving correctness of coordination as a first-class entity.

BIP is a language for the construction of concurrent systems by superposing three layers: behaviour, interaction and priorities. The layered approach of BIP separates concerns between interaction and computation. This is essential for component-based design of concurrent systems, because it allows global analysis of the coordination layer and reusability of written code.

Reo is a language for compositional specification of coordination protocols, i.e., protocols modeling the synchronization and dataflow among multiple components. These protocols consist of graph-like structures, called *connectors*. Reo connectors may compose together to form more complex connectors, allowing reusability and compositional construction of coordination protocols.

We provide a more detailed introduction to BIP and Reo in Section 2.

Motivation.   Both BIP and Reo advocate the necessity of separating coordination mechanisms from the coordinated components. In BIP one refers to this separation as the *architecture-based* design approach [11]. Reo literature uses the term *exogenous coordination* to describe the same fundamental principle [3, 4, 20]. Despite this fundamental agreement, the design choices underlying BIP and Reo differ. For example, BIP uses stateless interactions, while Reo allows stateful connectors. Establishing a formal relation between BIP and Reo is necessary to discover fundamental principles that drive the design of coordination languages.

Translations exist between numerous other coordination models and BIP and Reo, individually [12, 13, 21, 22]. Hence, a formal relationship between BIP and Reo yields insight, albeit indirect, into the relation of each with a wider range of related work.

Furthermore, establishing a formal relationship between BIP and Reo enables encodings that allow each of the two frameworks to benefit from tools and theoretical results obtained for the other. These toolchains include tools for editing, code generation, and model checking. We refer to [1] and [2, 4] for details.

Contributions.   We relate the most important semantic models of BI(P)[1] (i.e., BIP without the priority layer) and Reo. For Reo we consider *port automata* and *constraint automata*, which model Reo connectors at different levels of abstraction [16]. For BI(P) we consider *BIP architectures* [6] and *BIP interaction models*, i.e., sets of simple interaction expressions [11].

First, we provide a short summary of BIP and Reo in Section 2. Then, in Section 3, we define mappings between port automata and BIP architectures, and show that these distribute over composition modulo semantic equivalence. Hence, it is possible to compute these translations incrementally, in order to speed them up. In Section 4, we define mappings between stateless constraint automata and BIP interaction models. We show that all transformations preserve all properties of observable dataflow, which, for example, enables one to transfer safety properties established for some generated code, or the results of model checking from one model to the other. These mappings in the data-sensitive domain do not distribute over composition, but in Section 5 we briefly discuss a different translation scheme that still allows incremental translation. There, we discuss also the differences and similarities between BI(P) and Reo and other coordination languages, and point out future work.

Related Work.   Other authors have related and compared both BIP and Reo to other coordination languages. Bruni et al. encode BIP models into Petri nets [12], and Chkouri et al. present a translation of AADL into BIP [13]. Proença and Clarke provide a detailed comparison between Orc and Reo [21]. Arbab et al. provide a translation of Reo connectors into the Tile Model [5]. Krause compared Reo to Petri nets [18]. Talcott, Sirjani and Ren connect both ARC and PBRD to Reo by providing mappings between their semantic models [22].

Although an indirect comparison of BIP and Reo through their respective comparisons with other models, e.g., Petri nets, is certainly possible, the direct and formal translations we present in this paper allows direct translation tools between BIP and Reo, that are otherwise difficult, if not impossible, to construct based on such indirect comparisons.

2   Overview of BIP and Reo

2.1   BIP

A BIP system consist of a superposition of three layers: Behaviour, Interaction, and Priority. The behaviour layer encapsulates all computation, consisting of *atomic components* processing sequential code. *Ports* form the interface of a component through which it interacts with other components. BIP represents these atomic components as *Labeled Transition Systems* (LTS) having transitions labeled with ports and extended with data stored in local variables. The second layer defines component coordination by means of *BIP interaction models* [11]. For each *interaction* among components in a BIP system, the interaction model of that system specifies the set of ports synchronized by that interaction and the way

---

[1]Although BIP's notion of priority is equally applicable to the constraint automata semantics of Reo, Reo provides no syntax to specify such global priority preferences. Reo does have a weaker priority mechanism to specify local preferences by means of context sensitive channel `LossySync`, that prefers locally maximal dataflow.
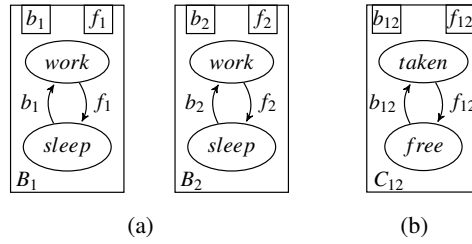
Figure 1: BIP components (a); coordinator (b).

data is retrieved, filtered and updated in each of the participating components. In the third layer, priorities impose scheduling constraints to resolve conflicts in case alternative interactions are possible. In the rest of this paper, we disregard priorities and focus mainly on interaction models (cf., footnote 1).

Data-agnostic semantics.   We first introduce a data-agnostic semantics for BIP.

Definition 1 (BIP component [6])**.** A *BIP component C* over a set of ports P_C is a labeled transition system $(Q, q^0, P_C, \rightarrow)$ over the alphabet 2^{P_C}. If $\mathscr{C}$ is a set of components, we say that $\mathscr{C}$ is *disconnected* iff $P_C \cap P_{C'} = \emptyset$ for all distinct $C, C' \in \mathscr{C}$. Furthermore, we define $P_{\mathscr{C}} = \bigcup_{C \in \mathscr{C}} P_C$.

Then, BIP defines an *interaction model* over a set of ports *P* to be a set of subsets of *P*. Interaction models are used to define synchronisations among components, which can be intuitively described as follows. Given a disconnected set of BIP components $\mathscr{C}$ and an interaction model γ over $P_{\mathscr{C}}$, the state space of the corresponding *composite component* $\gamma(\mathscr{C})$ is the cross product of the state spaces of the components in $\mathscr{C}$; $\gamma(\mathscr{C})$ can make a transition labelled by an interaction $N \in \gamma$ iff all the involved components (those that have ports in *N*) can make the corresponding transitions. A straightforward formal presentation can be found in [10] (cf., Definition 3 below). Thus, BIP interaction models are *stateless*: every interaction in γ is always allowed; it is enabled if all ports in the interaction are ready. However, [6] shows the need for statefull interaction, which motivates *BIP architectures*

Definition 2 (BIP architecture [6])**.** A *BIP architecture* is a tuple $A = (\mathscr{C}, P_A, \gamma)$, where $\mathscr{C}$ is a finite disconnected set of *coordinating* BIP components, P_A is a set of ports, such that $P_{\mathscr{C}} = \bigcup_{C \in \mathscr{C}} P_C \subseteq P_A$, and $\gamma \subseteq 2^{P_A}$ is a *data-agnostic interaction model*. We call ports in $P_A \setminus P_{\mathscr{C}}$ *dangling ports* of *A*.

Essentially, a BIP architecture is a structured way of combining an interaction model γ with a set of distinguished components, whose only purpose is to control which interactions in γ are applicable at which point in time (which depends on the states of the coordinating components).

Definition 3 (BIP architecture application [6])**.** Let $A = (\mathscr{C}, P_A, \gamma)$ be a BIP architecture, and $\mathscr{B}$ a set of components, such that $\mathscr{B} \cup \mathscr{C}$ is finite and disconnected, and that $P_A \subseteq P_{\mathscr{B}} \cup P_{\mathscr{C}}$. Write $\mathscr{B} \cup \mathscr{C} = \{B_i \mid i \in I\}$, with $B_i = (Q_i, q_i^0, P_i, \rightarrow_i)$. Then, the *application* $A(\mathscr{B})$ of *A* to $\mathscr{B}$ is the BIP component $(\prod_{i \in I} Q_i, (q_i)_{i \in I}, P_{\mathscr{B}} \cup P_{\mathscr{C}}, \rightarrow)$, where $\rightarrow$ is the smallest relation satisfying: $(q_i)_{i \in I} \xrightarrow{N} (q_i')_{i \in I}$ whenever

  1. $N = \emptyset$, and there exists an $i \in I$ such that $q_i \xrightarrow{\emptyset}_i q_i'$ and $q_j' = q_j$ for all $j \in I \setminus \{i\}$; or

  2. $N \cap P_A \in \gamma$, and for all $i \in I$ we have $N \cap P_i \neq \emptyset$ implies $q_i \xrightarrow{N \cap P_i}_i q_i'$, and $N \cap P_i = \emptyset$ implies $q_i' = q_i$.

The application $A(\mathscr{B})$, of a BIP architecture *A* to a set of BIP components $\mathscr{B}$, enforces coordination constraints specified by that architecture on those components [6]. The *interface* P_A of *A* contains all ports $P_{\mathscr{C}}$ of the coordinating components $\mathscr{C}$ and some additional ports, which must belong to the components

in $\mathscr{B}$. In the application $A(\mathscr{B})$, the ports belonging to P_A can participate only in interactions defined by the interaction model γ of A. Ports that do not belong to P_A are not restricted and can participate in any interaction.

Intuitively, an architecture can also be viewed as an incomplete system: the application of an architecture consists in "attaching" its dangling ports to the operand components. The operational semantics is that of composing all components (operands and coordinators) with the interaction model as described in the previous paragraph. The intuition behind transitions labelled by $\emptyset$ is that they represent *observable idling* (as opposed to internal transitions). This allows us to "desynchronise" combined architectures (see Definition 4) in a simple manner, since coordinators of one architecture can idle, while those of another performs a transition. Note that, if $N = \emptyset$, in item 2 of Definition 3, $N \cap P_i = \emptyset$, hence also, $q_i' = q_i$, for all i. Thus, intuitively, one can say that none of the components moves. Item 1, however, does allow one component to make a real move labelled by $\emptyset$, if such a move exists. Thus, the transitions labelled by $\emptyset$ interleave, reflecting the idea that in BIP synchronisation can happen only through ports.

Example 1 (Mutual exclusion [6]). Consider the components B_1 and B_2 in Figure 1(a). In order to ensure mutual exclusion of their `work` states, we apply the BIP architecture $A_{12} = (\{C_{12}\}, P_{12}, \gamma_{12})$, where C_{12} is shown in Figure 1(b), $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$ and $\gamma_{12} = \{\emptyset, \{b_1, b_{12}\}, \{b_2, b_{12}\}, \{f_1, f_{12}\}, \{f_2, f_{12}\}\}$. The interface P_{12} of A_{12} covers all ports of B_1, B_2 and C_{12}. Hence, the only possible interactions are those that explicitly belong to γ_{12}. Assuming that the initial states of B_1 and B_2 are `sleep`, and that of C_{12} is `free`, neither of the two states $(\texttt{free}, \texttt{work}, \texttt{work})$ and $(\texttt{taken}, \texttt{work}, \texttt{work})$ is reachable, i.e. the mutual exclusion property $(q_1 \neq \texttt{work}) \vee (q_2 \neq \texttt{work})$—where q_1 and q_2 are state variables of B_1 and B_2 respectively—holds in $A_{12}(B_1, B_2)$. $\triangle$

Definition 4 (Composition of BIP architectures [6]). Let $A_1 = (\mathscr{C}_1, P_1, \gamma_1)$ and $A_2 = (\mathscr{C}_2, P_2, \gamma_2)$ be two BIP architectures. Recall that $P_{\mathscr{C}_i} = \bigcup_{C \in \mathscr{C}_i} P_C$, for $i = 1, 2$. If $P_{\mathscr{C}_1} \cap P_{\mathscr{C}_2} = \emptyset$, then $A_1 \oplus A_2$ is given by $(\mathscr{C}_1 \cup \mathscr{C}_2, P_1 \cup P_2, \gamma_{12})$, where $\gamma_{12} = \{N \subseteq P_1 \cup P_2 \mid N \cap P_i \in \gamma_i, \text{ for } i = 1, 2\}$. In other words, γ_{12} is the interaction model defined by the conjunction of the characteristic predicates of γ_1 and γ_2.

Data-aware semantics.   Recently, the data-agnostic formalization of BIP interaction models was extended with data transfer, using the notion of *interaction expressions* [11]. Let $\mathscr{P}$ be a global set of ports. For each port $p \in \mathscr{P}$, let $x_p : D_p$ be a typed variable used for the data exchange at that port. For a set of ports $P \subseteq \mathscr{P}$, let $X_P = (x_p)_{p \in P}$. An interaction expression models the effect of an interaction among ports in terms of the data exchanged through their corresponding variables.

Definition 5 (Interaction expression [11]). An *interaction expression* is an expression of the form

$$(P \leftarrow Q).[g(X_Q, X_L) : (X_P, X_L) := up(X_Q, X_L) \mathbin{/\!/} (X_Q, X_L) := dn(X_P, X_L)],$$

where $P, Q \subseteq \mathscr{P}$ are *top* and *bottom* sets of ports; $L \subseteq \mathscr{P}$ is a set of *local* variables; $g(X_Q, X_L)$ is the boolean *guard*; $up(X_Q, X_L)$ and $dn(X_P, X_L)$ are respectively the *up-* and *downward data transfer* expressions.

For an interaction expression α as above, we define by $top(\alpha) \stackrel{\Delta}{=} P$, $bot(\alpha) \stackrel{\Delta}{=} Q$ and $supp(\alpha) \stackrel{\Delta}{=} P \cup Q$ the sets of top, bottom and all ports in α, respectively. We denote g_α, up_α and dn_α the guard, upward and downward transfer corresponding expressions in α.

The first part of an interaction expression, $(P \leftarrow Q)$, describes the control flow as a dependency relation between the bottom and the top ports. The expression in the brackets describes the data flow, first "upward"—from bottom to top ports—and then "downward". The guard $g(X_Q, X_L)$ relates these two parts: interaction is enabled only when the values of the local variables together with those of variables

associated to the bottom ports satisfy a boolean condition. As a side effect, an interaction expression may also modify local variables in X_L. Intuitively, such an interaction expression can *fire* only if its guard is true. When it fires, its upstream transfer is computed first using the values offered by its participating BIP components. Then, the downstream transfer modifies all the port variables with updated values.

Definition 6 (BIP interaction models [11])**.** A *(data-aware) BIP interaction model* is a set Γ of *simple BIP connectors* α, which are BIP interaction expressions of the form

$$(\{w\} \leftarrow A).[g(X_A): (x_w, X_L) := up(X_A) \,/\!/\, X_A := dn(x_w, X_L)],$$

where $w \in P$ is a single top port, $A \subseteq P$ is a set of ports, such that $w \notin A$, and neither *up* nor *g* involves local variables.

Example 2 (Maximum)**.** Let $\mathscr{P} = \{a, b, w, l\}$ be a set of ports of type integer, i.e., $x_p : D_p = \mathbb{Z}$, for all $p \in \mathscr{P}$, and consider the interaction expression (simple BIP connector)

$$\alpha_{\max} = (\{w\} \leftarrow \{a, b\}).[\mathtt{tt}: x_l := \max(x_a, x_b) \,/\!/\, x_a, x_b := x_l],$$

where $\mathtt{tt}$ is true. First, the connector takes the values presented at ports a and b. Then, the simple BIP connector $\alpha_{\max}$ computes atomically the maximum of x_a and x_b and assigns it to its local variable x_l. Finally, $\alpha_{\max}$ assigns atomically the value of x_l to both x_a and x_b. $\triangle$

BIP interaction expressions capture complete information about all aspects of component interaction—i.e. synchronisation and data transfer possibilities—in a structured and concise manner. Thus, by examining interaction expressions, one can easily understand, on the one hand, the interaction model used to compose components and, on the other hand, how the valuations of data variables affect the enabledness of the interactions and how these valuations are modified. Furthermore, a formal definition of a composition operator on interaction expressions is provided in [11], which allows combining such expressions hierarchically to manage the complexity of systems under design. Since any BIP system can be flattened, this hierarchical composition of interaction expressions is not relevant for the semantic comparison of BIP and Reo in this paper. Nevertheless, the possibility of concisely capturing all aspects of component interaction in one place is rather convenient.

2.2 Reo

Reo is a coordination language wherein graph like structures express concurrency constraints (e.g., synchronization, exclusion, ordering, etc.) among multiple components. These structures consist of a composition of channels and nodes, collectively called *connectors* or *circuits*. A channel in Reo has exactly two *ends*, and each end either accepts data items, if it is a *source end*, or offers data items, if it is a *sink end*. Moreover, a channel has a *type* for its behaviour in terms of a formal constraint on the dataflow through its two ends. Its abstract definition of channels and its notion of channel types make Reo an extensible programming language. Beside the established channel types (Table 1 contains some of them) Reo allows arbitrary user-defined channel types.

Multiple ends may glue together into *nodes* with a fixed *merge-replicate* behaviour: a data item out of a single sink end coincident on a node, atomically propagates to all source ends coincident on that node. This propagation happens only if all their respective channels allow the data exchange. A node is called a *source node* if it consists of source ends, a *sink node* if it consists of sink ends, and a *mixed node* otherwise. Together, the source and sink nodes of a connector constitute its set of *boundary nodes/ports*.
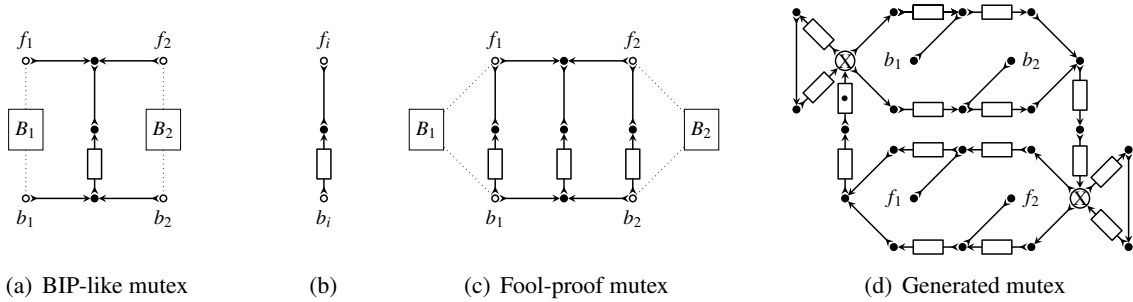
Figure 2: Fool-proof (c) mutual exclusion protocol in Reo, composed from a BIP-like (a) mutual exclusion connector and an altenator connector (b), and the generated Reo circuit (d) from Example 5.

Example 3. Figure 2(a) shows a Reo connector that achieves mutual exclusion of components B_1 and B_2, exactly as the BIP system shown in Figure 1 does. This connector consists of a composition of channels and nodes in Table 1. The Reo connector atomically accepts data from either b_1 or b_2 and puts it into the `FIFO1` channel, a buffer of size one. A full `FIFO1` channel means that B_1 or B_2 holds the lock. If one of the components writes to f_1 or f_2, the `SyncDrain` channel flushes the buffer, and the lock is released, returning the connector to its initial configuration, where B_1 and B_2 can again compete for exclusive access by attempting to write to b_1 or b_2.

Note that this connector is not fool-proof. Even if B_1 takes the lock, B_2 may release it, and vice versa. Hence, exactly as the BIP architecture in Figure 1, the Reo connector in Figure 2(a) relies on the conformance of the coordinated components B_1 and B_2. The expected behaviour of B_i, $i = 1, 2$, is that it alternates writes on the b_i and f_i, and that every write on f_i comes after a write on b_i. Depending on such assumptions may not be ideal. The connector, shown in Figure 2(b), makes this expected behaviour explicit. By composing two such connectors with the connector in Figure 2(a), we obtain a fool-proof mutual exclusion protocol, as shown in Figure 2(c). Figure 4(c) shows the constraint automaton semantics of the connector in Figure 2(c). Unlike the case of the connector in Figure 2(a) or the BIP architecture in Figure 1, non-compliant writes to b_i or f_i ports of the connector in Figure 2(c) will *block* component B_i, but cannot *break* the mutual exclusion protocol that this connector implements.   △
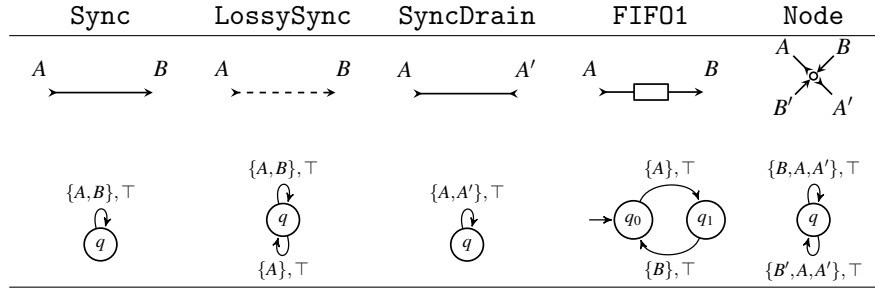
Formal semantics of Reo. Reo has a variety of formal semantics [4, 16]. In this paper we use its operational *constraint automaton* (CA) semantics [8].

Definition 7 (Constraint automata [8]). Let $\mathcal{N}$ be a set of nodes and $\mathcal{D}$ a set of data items. A data constraint is a formula in the language of the grammar

$$g \rightarrow \top \mid \neg g \mid g \wedge g \mid \exists d_p(g) \mid d_p = v, \quad \text{with } p \in \mathcal{N}, v \in \mathcal{D},$$

where variable d_p represents the data assigned to (i.e., exchanged through) port p. Let $\models$ denote the obvious satisfaction relation between data constraints and data assignments $\delta : N \rightarrow \mathcal{D}$, with $N \subseteq \mathcal{N}$, and write $DC(\mathcal{N}, \mathcal{D})$ for the set of all data constraints. A constraint automaton (over data domain $\mathcal{D}$) is a tuple $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0)$ where Q is a set of states, $\mathcal{N}$ is a finite set of nodes, $\rightarrow \subseteq Q \times 2^{\mathcal{N}} \times DC(\mathcal{N}, \mathcal{D}) \times Q$ is a transition relation, and $q_0 \in Q$ is the initial state.

In this paper, we consider only finite data domains, although most of our results generalize to infinite data domains. Over a finite data domain, the data constraint language $DC(\mathcal{N}, \mathcal{D})$ is expressive enough

Sync	LossySync	SyncDrain	FIFO1	Node

Table 1: Some primitives in the Reo language with CA semantics over a singleton data domain $\mathscr{D}$.

to define any data assignment. For notational convenience, we relax, in this paper, the definition of data constraints and allow the use of set-membership and functions in the data constraints. However, we preserve the intention that a data constraint describes a set of data assignments.

Table 1 shows the CA semantics for some typical Reo primitives. The CA semantics of every Reo connector can be derived as a composition of the constraint automata of its primitives, using the CA product operation in Definition 8. On the other hand, every constraint automaton (over a finite data domain) translates back into a Reo connector [7]. Because of this correspondence, we may consider Reo and CA as equivalent, and focus on constraint automata only.

If a constraint automaton $\mathscr{A}$ has only one state, $\mathscr{A}$ is called *stateless*. If the data domain $\mathscr{D}$ of $\mathscr{A}$ is a singleton, $\mathscr{A}$ is called a *port automaton* [17]. In that case, we omit data constraints, because all satisfiable constraints reduce to $\top$.

Definition 8 (Product of CA [8])**.** Let $\mathscr{A}_i = (Q_i, \mathscr{N}_i, \rightarrow_i, q_{0,i})$ be a constraint automaton, for $i = 1, 2$. Then the product $\mathscr{A}_1 \bowtie \mathscr{A}_2$ of these automata is the automaton $(Q_1 \times Q_2, \mathscr{N}_1 \cup \mathscr{N}_2, \rightarrow, (q_{0,1}, q_{0,2}))$, whose transition relation is the smallest relation obtained by the rule: $(q_1, q_2) \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} (q_1', q_2')$ whenever

1. $q_1 \xrightarrow{N_1, g_1}_1 q_1'$, $q_2 \xrightarrow{N_2, g_2}_2 q_2'$, and $N_1 \cap \mathscr{N}_2 = N_2 \cap \mathscr{N}_1$, or

2. $q_i \xrightarrow{N_i, g_i}_i q_i'$, $N_j = \emptyset$, $g_j = \top$, $q_j' = q_j$, and $N_i \cap \mathscr{N}_j = \emptyset$ with $j \in \{1, 2\} \setminus \{i\}$.

It is not hard to see that constraint automata product operator is associative and commutative modulo equivalence of state names and data constraints.

Definition 9 (Hiding in CA [8])**.** Let $\mathscr{A} = (Q, \mathscr{N}, \rightarrow, q_0)$ be a constraint automaton, and $P = \{p_1, \ldots, p_n\}$ a set of nodes. Then hiding nodes P of $\mathscr{A}$ yields an automaton $\exists P(\mathscr{A}) = (Q, \mathscr{N} \setminus P, \rightarrow_\exists, q_0)$, where $\rightarrow_\exists$ is given by $\{(q, N \setminus P, \exists d_{p_1} \cdots \exists d_{p_n}(g), q') \mid (q, N, g, q') \in \rightarrow\}$.

The hiding operator affects only transition labels, and preserves the structure of the automaton. Hence the hiding operator offers a technique to alter the interface of a component or connector without modifying its behaviour. As hiding of non-shared nodes distributes over the product, hiding of non-shared nodes commutes with constraint automata product.

Example 4 (Product and hide)**.** Consider the Reo connectors in Figure 2. Using Definition 8, and the primitive constraint automata from Table 1, we find their CA semantics as shown in Figures 4(a), 4(b), and 4(c), respectively. If we compute the product of the automaton $\mathscr{A}_0$ in Figure 4(a) with the automata $\mathscr{A}_i$, $i = 1, 2$, in Figure 4(b), then we obtain an automaton $\mathscr{A}$, whose part reachable from the initial state $(0, 0, 0)$ is shown in Figure 4(c). $\triangle$

Figure 3:

Reo		BIP
[7] ↕ [8]	bip₁	‖ [6]
PA ⟷	reo₁	Arch
f₁ ↘	LTS	↙ g₁

(a) data-agnostic domain

Reo		BIP
[7] ↕ [8]	bip₂	‖ [11]
CA± ⟷	reo₂	IM
f₂ ↘	LTS	↙ g₂

(b) data-sensitive domain

Figure 3: Translations and interpretations in data-agnostic and data-sensitive domain.

3   Port automata and BIP architectures

To study the relation between BIP and Reo with respect to synchronization, we start by defining a correspondence between them in the data-agnostic domain. This correspondence consists of a pair of mappings between the sets containing semantic models of BIP and Reo connectors. For the data independent semantic model of Reo connectors we choose port automata: a restriction of constraint automata over a singleton set as data domain. We model BIP connectors by BIP architectures introduced in [6]. In order to compare the behaviour of BIP and Reo connectors we interpret them as labeled transition systems. We define a mapping reo_1 that transforms BIP architectures into port automata, and a mapping bip_1 that transforms port automata into BIP architectures. We then show that these mappings preserve (1) properties closed under bisimulation, and (2) composition structure modulo semantic equivalence.

3.1   Interpretation of BIP and Reo

To compare the behaviour of BIP and Reo connectors, we interpret all connectors as labeled transitions systems with one initial state and an alphabet 2^P, for a set of ports P. We write LTS for the class of all such labeled transition systems.

   Figure 3(a) shows our translations and interpretations. The objects PA, Arch and LTS are, respectively, the classes of port automata, BIP architectures, and labeled transition systems. The mappings bip_1, reo_1, f_1 and g_1, respectively, translate Reo to BIP, BIP to Reo, Reo to LTS, and BIP to LTS.

   We first consider the semantics of connectors. Since BIP connectors differ internally from Reo connectors, we restrict our interpretation to their observable behaviour. This means that we hide the ports of the coordinating components in BIP architectures. For port automata this means that for our comparison, we implicitly assume that all names represent boundary nodes.

   The interpretation of a port automaton in LTS is defined by

$$\mathsf{f}_1((Q, \mathscr{N}, \rightarrow, q_0)) = (Q, 2^{\mathscr{N}}, \rightarrow, q_0). \tag{1}$$

Hence f_1 acts essentially as an identity function, justifying our choice of interpretation. Next, we define the interpretation of BIP architectures using their operational semantics obtained by applying them on dummy components and hiding all internal ports. Let $A = (\mathscr{C}, P, \gamma)$ be a BIP architecture with coordinating components $\mathscr{C} = \{C_1, \ldots, C_n\}$, $n \geq 0$, and $C_i = (Q_i, q_i^0, P_i, \rightarrow_i)$. Recall that $P_{\mathscr{C}} = \bigcup_i P_i$ is the set of internal ports in A. Define $D = (\{q_D\}, q_D, P, \{(q_D, N, q_D) \mid \emptyset \neq N \subseteq P \setminus P_{\mathscr{C}}\})$ as a dummy component relative to the BIP architecture A. Using Definition 3, we compute the BIP architecture application $A(\{D\}) = ((\prod_{i=1}^n Q_i) \times \{q_D\}, (\mathbf{q}^0, q_D), P, \rightarrow_s)$ of A to its dummy component D. Then,

$$\mathsf{g}_1(A) = (\textstyle\prod_{i=1}^n Q_i \times \{q_D\}, 2^{P \setminus P_{\mathscr{C}}}, \{((\mathbf{q}, q_D), N \setminus P_{\mathscr{C}}, (\mathbf{q}', q_D)) \mid (\mathbf{q}, q_D) \xrightarrow{N}_s (\mathbf{q}', q_D)\}, (\mathbf{q}^0, q_D)) \tag{2}$$
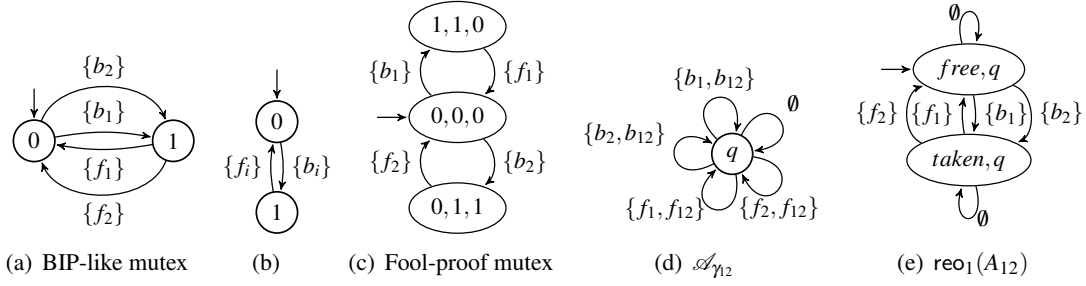
Figure 4: CA representations (a), (b), and (c) of Reo connectors Figures 2(a), 2(b), and 2(c), respectively; translation of the interaction model (d) and BIP architecture (e) of Figure 1.

In other words, $g_1(A)$ equals $A(\{D\})$ after hiding all internal ports $P_{\mathscr{C}}$. Note that we based our interpretation g_1 on the operational semantics of BIP architectures, i.e., BIP architecture application. This justifies the definition of interpretation of architectures.

Because of hiding, g_1 is not injective. Hence, our interpretation of BIP architectures induces a nontrivial equivalence given by equality of interpretations. In the sequel, we use a slightly stronger version of equivalence based on bisimulation [19].

Definition 10 (Bisimulation [19]). If $L_i = (Q_i, 2^{P_i}, \rightarrow_i, q_i^0) \in \text{LTS}$, $i = 1, 2$, then L_1 and L_2 are *bisimilar* $(L_1 \cong L_2)$ iff $P_1 = P_2$ and there exists $R \subseteq Q_1 \times Q_2$ such that $(q_1^0, q_2^0) \in R$, and $(q_1, q_2) \in R$ implies, for all $N \in 2^{P_i}$, $i, j \in \{1, 2\}$ with $i \neq j$, if $q_i \xrightarrow{N}_i q_i'$, then, for some q_j', $q_j \xrightarrow{N}_j q_j'$ and $(q_1', q_2') \in R$.

Definition 11 (Semantic equivalence). Let $\mathscr{A}, \mathscr{B} \in \text{PA}$ be port automata and $A, B \in \text{Arch}$ be BIP architectures. Then, $\mathscr{A}$ and $\mathscr{B}$ are *semantically equivalent* $(\mathscr{A} \sim \mathscr{B})$ iff $f_1(\mathscr{A}) \cong f_1(\mathscr{B})$, and A and B are *semantically equivalent* $(A \sim B)$ iff $g_1(A) \cong g_1(B)$.

With a common semantics for BIP and Reo, we can define the notion of preservation of properties expressible in this common semantics. Recall that a property of labeled transition systems corresponds to the subset of labeled transition systems satisfying that property.

Definition 12. Let $P \subseteq \text{LTS}$ be a property. Then, bip_1 *preserves* P iff $f_1(\mathscr{A}) \in P \Leftrightarrow g_1(\text{bip}_1(\mathscr{A})) \in P$ for all $\mathscr{A} \in \text{PA}$. Similarly, reo_1 *preserves* P iff $g_1(A) \in P \Leftrightarrow f_1(\text{reo}_1(A)) \in P$ for all $A \in \text{Arch}$.

3.2 BIP to Reo

To translate BIP connectors to Reo connectors, we first determine what elements of BIP architectures correspond to Reo connectors. Our interpretations of port automata and BIP architectures show that dangling ports in BIP architectures correspond to boundary port names in port automata. Furthermore, the mutual exclusion of the interactions in an interaction model in a BIP architecture simulates mutually exclusive firing of transitions in port automata. The definition of a coordinating component in a BIP architecture is almost identical to that of a port automaton, yielding an obvious translation.

Let $A = (\mathscr{C}, P, \gamma)$ be a BIP architecture, with $\mathscr{C} = \{C_1, \ldots, C_n\}$. Each C_i corresponds trivially to a port automaton $\widetilde{C}_i$. Let $\mathscr{A}_\gamma = (\{q\}, P, \rightarrow, q)$ be the stateless port automaton over P with transition relation $\rightarrow$ defined by $\{(q, N, q) \mid N \in \gamma\}$. Then $\mathscr{A}_\gamma$ can be seen as the port automata encoding of the interaction model γ. Recall that $P_{\mathscr{C}} = \bigcup_{C \in \mathscr{C}} P_C$. The corresponding port automaton of A is given by

$$\text{reo}_1(A) = \exists P_{\mathscr{C}}(\widetilde{C_1} \bowtie \cdots \widetilde{C_n} \bowtie \mathscr{A}_\gamma). \tag{3}$$

Example 5. We translate the BIP architecture in Example 1 using (3). First, we transform γ_{12} into a port automaton $\mathscr{A}_{\gamma_{12}}$, shown in Figure 4(d). Then, we compute the product of $\mathscr{A}_{\gamma_{12}}$ with the coordinating component C_{12} to obtain the port automaton corresponding to the BIP architecture A_{12}, shown in Figure 4(e). As mentioned in section Section 2.2, we can transform the port automaton in Figure 4(e) into a Reo connector, using the method described in [7]. This mechanical translation yields the Reo connector in Figure 2(d). Here, the dot in the FIFO1 buffer indicates that its initial state is the full state. The crossed node represents an *exclusive router*, which atomically takes data from a coincident sink end, and provides it to a single coincident source end. Note that the port automaton semantics of the connector in Figure 2(a) (see Figure 4(a)) is similar to the automaton in Figure 4(e), up to empty transitions.   △

3.3   Reo to BIP

In BIP, interaction is memoryless. This means that a stateful channel in Reo must translate to a coordinating component. In fact, we may encode the whole Reo connector as one such component.

Let $\mathscr{A}_i$, $i = 1, 2$, be two port automata, and let $p \in \mathscr{N}_1 \cap \mathscr{N}_2$ be a shared port of $\mathscr{A}_1$ and $\mathscr{A}_2$. Suppose that we know how to translate $\mathscr{A}_i$ into a BIP architecture A_i. If p is not a dangling port of A_1, then, by symmetry, p is not a dangling port of A_2. But now, A_1 and A_2 are not composable, because there components are not disconnected. Hence, since we want the translation to preserve composition, p should be a dangling port.

Let $\mathscr{A} = (Q, \mathscr{N}, \rightarrow, q_0)$ be a port automaton. We construct a corresponding BIP architecture. Duplicate all ports in $\mathscr{N}$ by defining $N' = \{n' \mid n \in N\}$ for all $N \subseteq \mathscr{N}$. We do not use a port n', for $n \in \mathscr{N}$, for composition. Their exact name is therefore not important, but merely their relation to its dangling brother n. Trivially, $\overline{\mathscr{A}} = (Q, q_0, \mathscr{N}', \rightarrow_c)$, with $\rightarrow_c = \{(q, N', q') \mid (q, N, q') \in \rightarrow\}$, is a BIP component (cf., Definition 1). Essentially, $\mathscr{A}$ and $\overline{\mathscr{A}}$ are the same labeled transition system. Now we define:

$$\mathsf{bip}_1(\mathscr{A}) = (\{\overline{\mathscr{A}}\}, \mathscr{N} \cup \mathscr{N}', \{N \cup N' \mid N \subseteq \mathscr{N}\}). \tag{4}$$

Thus, bip_1 uses the port automaton as the coordinating component of the generated BIP architecture.

Example 6. Let $\mathscr{A}$ be the port automaton in Figure 4(b) over the name set $\mathscr{N} = \{b_i, f_i\}$. We determine $\mathsf{bip}_1(\mathscr{A})$. Obtain $\overline{\mathscr{A}}$ by adding adding a prime to each port in $\mathscr{A}$. The interaction model of $\mathsf{bip}_1(\mathscr{A})$ consist of $\{N \cup N' \mid N \subseteq \mathscr{N}\} = \{\emptyset, \{b_i, b_i'\}, \{f_i, f_i'\}, \{b_i, b_i', f_i, f_i'\}\}$. Hence, $\mathsf{bip}_1(\mathscr{A})$ is given by th BIP architecture $(\{\overline{\mathscr{A}}\}, \{b_i, f_i, b_i', f_i'\}, \{\emptyset, \{b_i, b_i'\}, \{f_i, f_i'\}, \{b_i, b_i', f_i, f_i'\}\})$.

3.4   Preservation of properties

To confirm that translations reo_1 and bip_1 preserve properties, we first investigate whether Figure 3(a) commutes, i.e., $\mathsf{f}_1(\mathsf{reo}_1(A)) = \mathsf{g}_1(A)$ and $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A})) = \mathsf{f}_1(\mathscr{A})$, for $A \in \mathrm{Arch}$ and $\mathscr{A} \in \mathrm{PA}$.

First, note that the equations $\mathsf{f}_1(\mathsf{reo}_1(A)) = \mathsf{g}_1(A)$ and $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A})) = \mathsf{f}_1(\mathscr{A})$ cannot hold, because their state spaces differ. For example, g_1 alters the state space by adding the state of a dummy component, and reo_1 adds the state of the port automaton encoding of the interaction model. Therefore we view these equations modulo bisimulation of labeled transition systems from Definition 10.

Next, consider the equation $\mathsf{f}_1(\mathsf{reo}_1(A)) \cong \mathsf{g}_1(A)$, for some BIP architecture $A = (\{C_1, \ldots, C_n\}, P, \gamma)$. Suppose that two distinct coordination components C_i and C_j, $1 \leq i < j \leq n$, each contains an empty-labeled transition, i.e., there exist transistions $(q_i, \emptyset, q_i') \in \rightarrow_i$ and $(q_j, \emptyset, q_j') \in \rightarrow_j$. When we translate A to a port automaton using reo_1, the second rule in Definition 8 yields a *single* transition in $\mathsf{f}_1(\mathsf{reo}_1(A))$ from a global state where component C_i is in state q_i and C_j is in state q_j, to a global state where C_i is

in state q_i' and C_j is in state q_j'. However, BIP semantics does not allow independent progress of state-changing empty-labeled transitions, which means that this single transition exists only when $q_i' = q_i$ and $q_j' = q_j$. Indeed, the first rule of Definition 3 allows either C_i or C_j to change state, and the second rule implies $q_i' = q_i$ and $q_j' = q_j$ for $N = \emptyset$. Because of this, we need to exclude BIP architectures where two coordinating components can make a state-changing empty-labeled transition. Moreover, as we consider composition of BIP architectures in Section 3.5, we exclude BIP architectures containing a single coordinating component that can make a state-changing empty-labeled transition, and restrict Arch to $\text{Arch}' = \{A \in \text{Arch} \mid \forall C_i \in \mathscr{C} : q_i \xrightarrow{\emptyset}_i q_i' \Rightarrow q_i' = q_i\}$. Finally, consider the equation $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A})) \cong \mathsf{f}_1(\mathscr{A})$, for some port automaton $\mathscr{A}$. Note that the interaction model of $\mathsf{bip}_1(\mathscr{A})$ contains the empty set. Hence, the second rule in Definition 3 yields empty-labeled self-transitions in $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A}))$. Since f_1 acts like the identity, we conclude that $\mathscr{A}$ should have empty-labeled self-transitions, i.e., $q' = q$ implies $(q, \emptyset, q') \in \to$. On the other hand, suppose that $(q, \emptyset, q') \in \to$. Then the coordinating component of $\mathsf{bip}_1(\mathscr{A})$ should not contain a state-changing empty-labeled transition, hence $q' = q$. Therefore, we restrict PA to $\text{PA}' = \{\mathscr{A} \in \text{PA} \mid q \xrightarrow{\emptyset} q' \Leftrightarrow q' = q\}$.

Theorem 1. *For all $\mathscr{A} \in \text{PA}'$ and $A \in \text{Arch}'$ we have $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A})) \cong \mathsf{f}_1(\mathscr{A})$ and $\mathsf{f}_1(\mathsf{reo}_1(A)) \cong \mathsf{g}_1(A)$.*

Proof. Using Definition 3, Definition 8, $A \in \text{Arch}'$, $\mathscr{A} \in \text{PA}'$, and the fact that $(q_D, \emptyset, q_D) \notin \to_D$, it follows that (1) $\sim$ given by $(q, q_D) \sim q$ for all $q \in Q$ is a bisimulation between $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A}))$ and $\mathsf{f}_1(\mathscr{A})$, where Q is the state space of $\mathscr{A}$, and (2) $\approx$ given by $(\mathbf{q}, q_I) \approx (\mathbf{q}, q_D)$ for all $\mathbf{q} = (q_i)_{i \in I} \in \prod_{i \in I} Q_i$, is a bisimulation, where Q_i, $i \in I$, are the state spaces of the coordinating components of A. See [14] for a detailed proof. $\qquad\square$

Corollary 1. bip_1 *and* reo_1 *preserve all properties closed under bisimulation, i.e., for all $P \subseteq \text{LTS}$, $\mathscr{A} \in \text{PA}'$ and $A \in \text{Arch}'$ we have $\mathsf{f}_1(\mathscr{A}) \in P \Leftrightarrow \mathsf{g}_1(\mathsf{bip}_1(\mathscr{A})) \in P$ and $\mathsf{g}_1(A) \in P \Leftrightarrow \mathsf{f}_1(\mathsf{reo}_1(A)) \in P$.*

Example 7. Consider the following safety property φ satisfied by the Reo connector in Figure 2(c): "if b_1 fires, then b_2 fires only after f_1 fires". Clearly, the automaton $\mathscr{A}'$, obtained from Figure 4(c) by adding empty self-transitions, satisfies this property as well. Using Corollary 1, we conclude that the BIP architecture $\mathsf{bip}_1(\mathscr{A}) = \mathsf{bip}_1(\mathscr{A}')$ satisfies φ. More generally, Corollary 1 allows model checking of BIP architectures with Reo model checkers. $\qquad\triangle$

3.5 Compatibility with composition

BIP architectures and port automata have their own notions of composition. This raises the question of whether our translations preserve composition structures. We show that, under specific conditions, our translations preserve composition modulo semantic equivalence. Recall the port automaton representation of the interaction model (Section 3.2).

Lemma 1. *Let $A_i = (\mathscr{C}_i, P_i, \gamma_i) \in \text{Arch}$, $i = 1, 2$, with $P_{\mathscr{C}_1} \cap P_{\mathscr{C}_2} = \emptyset$ and $\emptyset \in \gamma_1 \cap \gamma_2$. Then, we have that $\mathscr{A}_{\gamma_{12}} \sim \mathscr{A}_{\gamma_1} \bowtie \mathscr{A}_{\gamma_2}$, where γ_{12} be the interaction model of $A_1 \oplus A_2$.*

Proof. Follows easily from Definition 8 and Definition 4. See [14] for a detailed proof. $\qquad\square$

Suppose that $\mathsf{reo}_1(A_1 \oplus A_2) \sim \mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2)$, for any two BIP architectures $A_1, A_2 \in \text{Arch}'$. Definition 8 implies $N_{\mathsf{reo}_1(A_1 \oplus A_2)} = N_{\mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2)} = N_{\mathsf{reo}_1(A_1)} \cup N_{\mathsf{reo}_1(A_2)}$. In other words, the name set of port automaton $\mathsf{reo}_1(A_1 \oplus A_2)$ is the union of the name set of the port automata $\mathsf{reo}_1(A_i)$, $i = 1, 2$. Hence, $N_{\mathsf{reo}_1(A_i)} \subseteq N_{\mathsf{reo}_1(A_1 \oplus A_2)}$, for $i = 1, 2$. This means that the dangling ports of $\mathsf{reo}_1(A_1 \oplus A_2)$ contain all dangling ports of $\mathsf{reo}_1(A_i)$. Therefore, we need to assume that $P_{\mathscr{C}_1} \cap P_2 = P_{\mathscr{C}_2} \cap P_1 = \emptyset$.

Note that this is only a mild assumption. Indeed, if $p \in P_{\mathscr{C}_1} \cap P_2$ is a dangling port of P_2, connected directly to a component in A_1. Then, we first add a (dangling) port x to A_1 and synchronize p with p' by considering the BIP interaction model $\gamma_1' = \{N \cup \{x\} \mid p \in N \in \gamma_1\} \cup \{N \mid p \notin N \in \gamma\}$. Finally, we rename p to x in A_2. The resulting architectures satisfy the assumption.

Theorem 2. $\mathsf{reo}_1(A_1 \oplus A_2) \sim \mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2)$ *for all* $A_i = (\mathscr{C}_i, P_i, \gamma_i) \in \mathrm{Arch}'$, *with* $P_{\mathscr{C}_1} \cap P_2 = P_{\mathscr{C}_2} \cap P_1 = \emptyset$ *and* $\emptyset \in \gamma_1 \cap \gamma_2$.

Proof. Let $\mathscr{C}_1 \cup \mathscr{C}_2 = \{C_1, \ldots, C_n, \ldots, C_m\}$, with $C_i \in \mathscr{C}_1$ iff $i \leq n$. By definition, we have $\mathsf{reo}_1(A_1 \oplus A_2) = \exists P_{\mathscr{C}_1 \cup \mathscr{C}_2}(\widetilde{C_1} \bowtie \cdots \widetilde{C_n} \bowtie \widetilde{C_{n+1}} \bowtie \cdots \widetilde{C_m} \bowtie \mathscr{A}_{\gamma_{12}})$. Next, we use the bisimulation of port automata (i.e., constraint automata with data contraint $\top$) as defined in [8]. Composition ($\bowtie$) of port automata is commutative and associative up to bisimulation [8]. Using Lemma 1, it follows that $\mathsf{reo}_1(A_1 \oplus A_2) \cong \exists P_{\mathscr{C}_1} \exists P_{\mathscr{C}_2}(\widetilde{C_1} \bowtie \cdots \widetilde{C_n} \bowtie \mathscr{A}_{\gamma_1} \bowtie \widetilde{C_{n+1}} \bowtie \cdots \widetilde{C_m} \bowtie \mathscr{A}_{\gamma_2})$. Indeed, since f_1 is like the identity, it follows that semantic equivalence $\sim$ coincides with bisimulation $\simeq$ of port automata as defined in [8]. Now, we use our assumption that $P_{\mathscr{C}_1} \cap P_2 = P_{\mathscr{C}_2} \cap P_1 = \emptyset$, and the fact that $\widetilde{C_1}, \ldots, \widetilde{C_n}$, and $\mathscr{A}_{\gamma_1}$ do not use ports from $P_{\mathscr{C}_2}$. Then, $\mathsf{reo}_1(A_1 \oplus A_2) \cong \exists P_{\mathscr{C}_1}(\widetilde{C_1} \bowtie \cdots \widetilde{C_n} \bowtie \mathscr{A}_{\gamma_1}) \bowtie \exists P_{\mathscr{C}_2}(\widetilde{C_{n+1}} \bowtie \cdots \widetilde{C_m} \bowtie \mathscr{A}_{\gamma_2}))$. We conclude that $\mathsf{reo}_1(A_1 \oplus A_2) \cong \mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2)$. Since, f_1 is like the identity, it is not hard to see that f_1 takes bisimilar port automata to bisimilar labeled transition systems. Therefore, reo_1 is a homomorphism up to semantic equivalence, i.e., $\mathsf{reo}_1(A_1 \oplus A_2) \sim \mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2)$. $\qquad \square$

Theorem 3. $\mathsf{bip}_1(\mathscr{A}_1 \bowtie \mathscr{A}_2) \sim \mathsf{bip}_1(\mathscr{A}_1) \oplus \mathsf{bip}_1(\mathscr{A}_2)$ *for all* $\mathscr{A}_i \in \mathrm{PA}'$.

Proof. Note that, since f_1 is like the identity, semantic equivalence $\sim$ coincides with bisimulation $\simeq$ of port automata [8]. As $\simeq$ is a congruence with respect to the composition $\bowtie$ of port automata, we conclude that $\sim$ is a congruence too (i.e., $\mathsf{f}_1(\mathscr{A}_i) \cong \mathsf{f}_1(\mathscr{A}_i')$, for $i = 1, 2$, implies $\mathsf{f}_1(\mathscr{A}_1 \bowtie \mathscr{A}_2) \cong \mathsf{f}_1(\mathscr{A}_1' \bowtie \mathscr{A}_2'))$.

Let $\mathscr{A}_i \in \mathrm{PA}'$, $i = 1, 2$, be two port automata. From Theorem 2, we conclude that $\mathsf{f}_1(\mathsf{reo}_1(A_1 \oplus A_2)) \cong \mathsf{f}_1(\mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2))$, for any $A_1, A_2 \in \mathrm{Arch}'$. Substitute $A_i = \mathsf{bip}_1(\mathscr{A}_i)$, for $i = 1, 2$. Then, $\mathsf{f}_1(\mathsf{reo}_1(\mathsf{bip}_1(\mathscr{A}_1) \oplus \mathsf{bip}_1(\mathscr{A}_2))) \cong \mathsf{f}_1(\mathsf{reo}_1(\mathsf{bip}_1(\mathscr{A}_1)) \bowtie \mathsf{reo}_1(\mathsf{bip}_1(\mathscr{A}_2)))$. Thus, $\mathsf{f}_1(\mathsf{reo}_1(\mathsf{bip}_1(\mathscr{A}_i))) \cong \mathsf{g}_1(\mathsf{bip}_1(\mathscr{A}_i)) \cong \mathsf{f}_1(\mathscr{A}_i)$, for $i = 1, 2$, by Theorem 1. Hence, using that $\sim$ is a congruence, we obtain $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A}_1) \oplus \mathsf{bip}_1(\mathscr{A}_2)) \cong \mathsf{f}_1(\mathscr{A}_1 \bowtie \mathscr{A}_2)$. Therefore, $\mathsf{g}_1(\mathsf{bip}_1(\mathscr{A}_1) \oplus \mathsf{bip}_1(\mathscr{A}_2)) \cong \mathsf{g}_1(\mathsf{bip}_1(\mathscr{A}_1 \bowtie \mathscr{A}_2))$. $\qquad \square$

Example 8. For any two ports x and y, let $\mathscr{A}_{\{x,y\}}$ be the port automaton of a synchronous channel (cf., Table 1), and let $C_{\{x,y\}}$ be its corresponding BIP component. Suppose we need to translate $\mathscr{A}_{\{a,b\}} \bowtie \mathscr{A}_{\{b,c\}}$ to a BIP architecture. Then we first compute $\mathsf{bip}_1(\mathscr{A}_{\{a,b\}}) = (\{C_{\{a',b'\}}\}, \{a, a', b, b'\}, \gamma_{\{a,b\}})$, with $\gamma_{\{a,b\}} = \{\emptyset, \{a, a'\}, \{b, b'\}, \{a, a', b, b'\}\}$. Next, we compute $\mathsf{bip}_1(\mathscr{A}_{\{b,c\}}) = (\{C_{\{b'',c''\}}\}, \{b, b'', c, c''\}, \gamma_{\{b,c\}})$, with $\gamma_{\{b,c\}} = \{\emptyset, \{b, b''\}, \{c, c''\}, \{b, b'', c, c''\}\}$. Note that we need to use a double prime now, because otherwise b' would be a shared port of $C_{\{a',b'\}}$ and $C_{\{b'',c''\}}$. Using Theorem 3, we find that $\mathsf{bip}_1(\mathscr{A}_{\{a,b\}} \bowtie \mathscr{A}_{\{b,c\}}) = \mathsf{bip}_1(\mathscr{A}_{\{a,b\}}) \oplus \mathsf{bip}_1(\mathscr{A}_{\{b,c\}}) = (\{C_{\{a',b'\}}, C_{\{b'',c''\}}\}, \{a, a', b, b', b'', c, c''\}, \gamma_{\{a,b,c\}})$, where $\gamma_{\{a,b,c\}}$ is the composition of $\gamma_{\{a,b\}}$ and $\gamma_{\{b,c\}}$.

Example 9. Consider the port automaton $\mathscr{A}'$, obtained from Figure 4(c) by adding empty self-transitions. If we translate $\mathscr{A}'$ to BIP, we obtain a BIP architecture $B_1 = \mathsf{bip}_1(\mathscr{A}')$, which has only a single coordinating component. From Example 4 we conclude $\mathscr{A}' \cong \mathscr{A}_0' \bowtie \mathscr{A}_1' \bowtie \mathscr{A}_2'$, where $\mathscr{A}_0$ is the port automaton in Figure 4(a), $\mathscr{A}_i$, $i = 1, 2$, is the port automaton in Figure 4(b), and $\mathscr{A}_i'$ is obtained from $\mathscr{A}_i$ by adding empty self-transitions. Now consider $B_3 = \mathsf{bip}_1(\mathscr{A}_0') \oplus \mathsf{bip}_1(\mathscr{A}_1') \oplus \mathsf{bip}_1(\mathscr{A}_2')$. Using Definition 4, we see that B_3 has three coordinating components. Nevertheless, Theorem 3 shows that B_3 is semantically equivalent to B. Therefore, Theorem 3 allows to compute translations compositionally. $\qquad \triangle$

4 Stateless CA's and interaction models

In Section 3 we established a correspondence between port automata and BIP architectures. Here, we offer translations between data-aware connector models in BIP and Reo.

First we determine the semantic model of the connectors. For BIP connectors we use BIP interaction models, i.e., sets of interaction expressions α, with a single top port that is not a bottom port, and whose guard and up functions are independent of local variables (Definition 5). We assume that every top port occurs only in one interaction expression per BIP interaction model. We denote the class of BIP interaction models by IM. For the semantics of Reo connectors we take a pair consisting of a constraint automaton together with a partition of its node set into source nodes $\mathscr{N}_{src}$, mixed nodes $\mathscr{N}_{mix}$, and sink nodes $\mathscr{N}_{snk}$. We call such pairs *constraint automata with polarity*. Due to the absence of coordinating components in the data sensitive model for BIP, we restrict ourselves here to stateless constraint automata, since BIP interaction expressions are stateless [6, 11]. We write $\text{CA}^{\pm}$ for the class of all stateless constraint automata with polarity, with $\mathscr{N}_{src} = P^* = \{p^* \mid p \in P\}$ and $\mathscr{N}_{snk} = P_* = \{p_* \mid p \in P\}$ for some set of ports P. This assumption is necessary to enable simulation of bidirectional ports in BIP. The reason we explicitly distinguish node types in this semantics is to give direction to dataflow, similar to BIP connectors. Usually such node type distinctions are implicit, but for preciseness we encode them as a partition within the semantics of Reo connectors.

As in Section 3, we interpret all connectors as labeled transition systems. Then we define translations between Reo connectors ($\text{CA}^{\pm}$) and BIP connectors (IM), and show that they preserve properties.

4.1 Interpretation of BIP and Reo

An important difference between BIP and Reo involves how they handle data. BIP uses bidirectional ports, while Reo treats input and output ports separately. Since the common semantics should support both approaches, we duplicate every bidirectional port of BIP to obtain two unidirectional ports, compatible with Reo. The sense of every reference to a bidirectional port in a BIP interaction expression maps that bidirectional port to its intended corresponding unidirectional port.

Let LTS be the class of all labeled transition systems over an alphabet $(D+1)^{2P}$, where D is a set of data items; $1 = \{0\}$ contains *void* or *null*, modeling the absence of data; and $2P$ is the *duplicated (unidirectional) port set* of a set of (bidirectional) ports P, that is, $2P = \{p^*, p_* \mid p \in P\}$. If data appears at p^* (i.e., $\delta(p^*) \neq 0$ for $\delta \in (D+1)^{2P}$), then we interpret this as input to the connector. If data appears at p_*, then we interpret this at output from the connector.

Consider Figure 3(b). Classes $\text{CA}^{\pm}$ and IM consist of constraint automata with polarity and interaction models. Morphisms bip_2 and reo_2 are translations of those classes and f_2 and g_2 are interpretations in a common LTS semantics. We do not intend to redefine the semantics of constraint automata with polarity and of interaction models in this section. Hence, we interpret them using their definitions from [8, 11].

We begin by defining the interpretation of stateless constraint automata with polarity. Given a stateless constraint automaton with polarity $\mathscr{A}$, we first determine the smallest set of bidirectional ports P such that $\mathscr{N}_{src}^{used} \subseteq P^*$ and $\mathscr{N}_{snk}^{used} \subseteq P_*$, where $\mathscr{N}_{src}^{used}$ and $\mathscr{N}_{snk}^{used}$ are all source and sink nodes that occur on a transition of $\mathscr{A}$. Then, we take $2P$ as the port names of $\text{f}_2(\mathscr{A})$. Finally, we obtain the transitions of $\text{f}_2(\mathscr{A})$ by replacing every transition labeled with N, g in $\mathscr{A}$ with a set of transitions labeled with $\delta \in \Delta(N, g)$, where $\Delta(N, g)$ contains all data assignments $\delta : 2P \to \mathscr{D} + 1$ that satisfy the data constraint N, g. We formalize this as follows. Let $\mathscr{A} = (\{q\}, \mathscr{N}_{src}, \mathscr{N}_{mix}, \mathscr{N}_{snk}, \to, q)$ be a stateless constraint automaton with polarity over a data domain $\mathscr{D}$. Define $\mathscr{N}_{src}^{used} = \bigcup\{N \cap \mathscr{N}_{src} \mid q \xrightarrow{N,g} q\}$, and

$\mathscr{N}_{snk}^{used} = \bigcup\{N \cap \mathscr{N}_{snk} \mid q \xrightarrow{N,g} q\}$. Let P be the smallest set, with $\mathscr{N}_{src}^{used} \subseteq P^*$ and $\mathscr{N}_{snk}^{used} \subseteq P_*$. Define

$$\mathsf{f}_2(\mathscr{A}) = (\{q\}, (\mathscr{D}+1)^{2P}, \{(q, \delta, q) \mid q \xrightarrow{N,g} q, \delta \in \Delta(N,g)\}), \tag{5}$$

where $\Delta(N,g) = \{\delta : 2P \to \mathscr{D}+1 \mid \delta(2P \setminus N) = \{0\}, \delta \models g\}$. Note that ports in $\mathscr{N}_{src} \setminus \mathscr{N}_{src}^{used}$ and $\mathscr{N}_{snk} \setminus \mathscr{N}_{snk}^{used}$ are important only for composition, which we do not consider in this paper.

Next, we interpret interaction models Γ by a single-state labeled transition system with labels describing all possible dataflows allowed by the guard, and up and down functions of some interaction expression in Γ. Before we provide a formal definition, we first introduce some notation. For every BIP interaction expression α, we write P_α for its bottom ports, g_α for its guard, up_w^α and up_L^α for the restriction of the up function to its top port and its local variables, respectively, and dn_{bot}^α for the restriction of the down function to its bottom ports. For every BIP interaction model Γ, we write $P_\Gamma = \bigcup_{\alpha \in \Gamma} P_\alpha$, and $D_\Gamma = \bigcup_{p \in P_\Gamma} \mathsf{D}_p$, where D_p is the data type of port p. For every data assignment $\delta : 2P_\Gamma \to D_\Gamma + 1$ we define $\delta_{up}(p) = \delta(p^*)$ and $\delta_{dn}(p) = \delta(p_*)$, for all $p \in P_\alpha$. Then, we define

$$\mathsf{g}_2(\Gamma) = (\{q\}, (D_\Gamma+1)^{2P_\Gamma}, \{(q, \delta, q) \mid \alpha \in \Gamma, \delta \in \Delta(\alpha) \subseteq (D_\Gamma+1)^{2P_\Gamma}\}), \tag{6}$$

where $\Delta(\alpha) = \{\delta \mid \delta(2P_\Gamma \setminus 2P_\alpha) = \{0\}, g_\alpha(\delta_{up}) = \mathtt{tt}, \delta_{dn} = dn_{bot}^\alpha(up_w^\alpha(\delta_{up}), up_L^\alpha(\delta_{up}))\}$. Note that we use the value of $up_w^\alpha(\delta_{up})$ as a local variable, since we consider only non-hierarchical interaction models.

In [11], Bliudze et al. encode BIP interaction models in *Top/Bottom components*, i.e., an automaton over interaction expressions together with local variables. Furthermore, they define a semantics for T/B components, which indirectly defines an interpretation of interaction models. Equation (6) imitates this interpretation without using Top/Bottom components explicitly.

Now that we defined the interpretation of our objects in LTS, we explore how these translations preserve properties that are expressible in LTS, as we did for their counterparts in Section 3.1.

4.2   Reo to BIP

Since BIP interaction models are stateless, we cannot translate an arbitrary constraint automaton (i.e., Reo connector) into BIP. Interaction models in BIP preclude keeping track of the state of a Reo connector. Hence, the translation of the interaction model of a BIP architecture into a port automaton in Section 3.2 inspires us for our translation bip$_2$.

Let $\mathscr{A}$ be a stateless constraint automaton over a data domain $\mathscr{D}$. Since we care only about external behaviour, we first hide all mixed nodes. Then, we transform every transition in $\mathscr{A}$ with label N, g into a simple BIP connector with N as its bottom ports, together with a guard, an up and a down function that mimic the data constraint g. We define the corresponding set bip$_2(\mathscr{A})$ of simple BIP connectors by the set of all transformed transitions from $\mathscr{A}$.

We first define the transformation of transitions into interaction expressions. For every label N, g in automaton $\mathscr{A}$, we define the simple BIP connector

$$\alpha(N,g) = (\{w_{N,g}\} \leftarrow P_N).[g_{src}(X_{src}) : Y_{snk} := \mathrm{solve}(g, X_{src}) \mathbin{/\!/} X_{snk} := Y_{snk}],$$

where P_N is the smallest set satisfying $N \cap \mathscr{N}_{bnd} \subseteq 2P_N$, g_{src} is any quantifier free formula equivalent to $\exists N \setminus \mathscr{N}_{src} : g$, the variables $Y_{snk} = \{y_p \mid p \in N \cap \mathscr{N}_{snk}\}$ are some fresh local variables, and $X_{src} = \{x_p \mid p \in N \cap \mathscr{N}_{src}\}$ and $X_{snk} = \{x_p \mid p \in N \cap \mathscr{N}_{snk}\}$ model the input and output values assigned to the bottom ports, and $\mathrm{solve}(g, X_{src})$ returns a vector Y_{snk} satisfying $\exists X_{mix} : g(X_{src}, X_{mix}, Y_{snk})$. All variables have data type $\mathscr{D}$ (the data domain of $\mathscr{A}$), i.e., $x_p : \mathscr{D}$ for all $p \in \mathscr{N}$. Note that the solve function in $\alpha(N,g)$ is not

deterministic. However, comparing the solve function to the random function in Figure 4 in [11], we see that this generality is justified. Now, we define bip_2 as follows:

$$\mathsf{bip}_2(\mathscr{A}) = \{\alpha(N,g) \mid (q,N,g,q) \in \rightarrow\}, \tag{7}$$

4.3 BIP to Reo

The correspondence between BIP interaction expressions and automata transitions from Section 4.2, provides the main idea for the translation of interaction models into stateless constraint automata. If Γ is a set of simple BIP connectors, we assign to every $\alpha \in \Gamma$ a transition τ_α labeled with $N(\alpha), g(\alpha)$, and subsequently construct the stateless constraint automaton consisting of all such τ_α transitions.

Let α be a simple BIP interaction expression. Recall our relaxation on the data constraint language in Section 2, and our notations regarding α in Section 4.1. Then, define $N(\alpha) \subseteq 2P_\alpha = \{p_*, p^* \mid p \in P_\alpha\}$ where $p^* \in N(\alpha)$ iff α assignes data to p in the upward data transfer, and $p_* \in N(\alpha)$ iff α assigns data to p in the downward data transfer. Furthermore, let $D_* = (d_{p_*})_{p \in P}$, $D^* = (d_{p^*})_{p \in P}$, and define

$$g(\alpha) = \bigwedge_{p \in P} d_{p^*}, d_{p_*} \in \mathsf{D}_p \wedge g_\alpha(D^*) \wedge D_* = dn^\alpha_{bot}(up^\alpha_w(D^*), up^\alpha_L(D^*)),$$

Note that $g(\alpha)$ is independent of the top port w, as we consider only non-hierarchical connectors.

Let Γ be a set of simple BIP connectors. Recall that $P_\Gamma = \bigcup_\alpha P_\alpha$ and $D_\Gamma = \bigcup_{p \in P_\Gamma} \mathsf{D}_p$. Then, define the constraint automaton $\mathsf{reo}_2(\Gamma)$ over D_Γ by

$$\mathsf{reo}_2(\Gamma) = (\{q\}, (P_\Gamma)^*, \emptyset, (P_\Gamma)_*, \{(q, N(\alpha), g(\alpha), q) \mid \alpha \in \Gamma\}, q). \tag{8}$$

Example 10. Consider the interaction expression $\alpha_{\max}$ from Example 2, with the data domains restricted to $\mathscr{D} = \{0, \ldots, 2^{32} - 1\}$. We translate the interaction model $\Gamma = \{\alpha_{\max}\}$ using (8), i.e., we compute $\mathscr{A} = \mathsf{reo}_2(\Gamma)$. Trivially, $\mathscr{A}$ is stateless. Its set of input ports equals $(P_\Gamma)^* = \{a^*, b^*\}$, and its set of output ports equals $(P_\Gamma)_* = \{a_*, b_*\}$. It has a unique transitions (q, N, g, q), with synchronization contraint $N = \{a^*, b^*, a_*, b_*\}$ and guard $g \equiv \bigvee_{x,y,z \in \mathscr{D} \,:\, z=\max(x,y)} (d_{a^*} = x \wedge d_{b^*} = y \wedge d_{a_*} = z \wedge d_{b_*} = z)$. △

4.4 Preservation of properties

To show the faithfulness of translations bip_2 and reo_2, we show that interpretations f_2 and g_2 commute with the translations bip_2 and reo_2 in Figure 3(b).

Theorem 4. *For all $\mathscr{A} \in \mathrm{CA}^\pm$ and all $\Gamma \in \mathrm{IM}$ we have $\mathsf{g}_2(\mathsf{bip}_2(\mathscr{A})) = \mathsf{f}_2(\mathscr{A})$ and $\mathsf{f}_2(\mathsf{reo}_2(\Gamma)) = \mathsf{g}_2(\Gamma)$.*

Proof. (Sketch). Let $\Gamma \in \mathrm{IM}$ and $\mathscr{A} \in \mathrm{CA}^\pm$. Then, $\Delta(\alpha(N,g)) = \Delta(N,g)$, and $\Delta(N(\alpha), g(\alpha)) = \Delta(\alpha)$, for all $\alpha \in \Gamma$, and all transition labels N, g in $\mathscr{A}$. From this and the definitions of f_2 and g_2, we see that $\mathsf{g}_2(\mathsf{bip}_2(\mathscr{A})) = \mathsf{f}_2(\mathscr{A})$, and $\mathsf{f}_2(\mathsf{reo}_2(\Gamma)) = \mathsf{g}_2(\Gamma)$, respectively. □

Corollary 2. *The translations bip_2 and reo_2 preserve all properties expressible in LTS, i.e., $\mathsf{f}_2(\mathscr{A}) \in P \Leftrightarrow \mathsf{g}_2(\mathsf{bip}_2(\mathscr{A})) \in P$ and $\mathsf{g}_2(\Gamma) \in P \Leftrightarrow \mathsf{f}_2(\mathsf{reo}_2(\Gamma)) \in P$ for all $P \subseteq \mathrm{LTS}$, $\mathscr{A} \in \mathrm{CA}^\pm$ and $\Gamma \in \mathrm{IM}$.*

Example 11. Consider the following safety property φ for the interaction expression $\alpha_{\max}$ from Example 2: "the value retrieved from port a equals zero". Clearly, this safety property does not hold, whenever a or b offers a non-zero integer. Note that φ depends solely on the interpretation of the interaction model $\Gamma = \{\alpha_{\max}\}$ in LTS, and hence φ is expressible in LTS. Using Corollary 2 we conclude that φ is false also for $\mathscr{A}_{\max} = \mathsf{reo}_2(\{\alpha_{\max}\})$. Thus, we know any executable code generated from the constraint automaton $\mathscr{A}_{\max}$ does not satisfy φ. More generally, Corollary 2 allows us to use the Reo compiler to generate correct code for a BIP interaction model. △

5   Conclusions and Future Work

BIP and Reo find common ground in their stimulation of exogenous system design. This means that they force the explicit modeling of coordination constraints. A clear and formal separation between coordination (connectors) and computation (components) allows the software architect to analyze the interaction of the components using automated tools. The exogenous approach of BIP and Reo contrasts with the endogenous approach supported in process algebra and other languages where coordination is woven into the code of the components. For example, process algebra does not supply constructs to enforce the separation of concerns necessary in exogenous coordination [20].

Multiparty synchronization constitutes a fundamental coordination concept in BIP (represented by interactions in a BIP interaction model) and Reo (represented by synchronization constraints in constraint automata). Our translations show that these representations of multiparty synchronization coincide.

The BIP framework concretely *defines* what separates computation (BIP behaviour) and coordination (BIP interaction), while Reo merely *separates* computation (Reo components) and coordination (Reo connector) structurally. Indeed, Reo does not force a fixed universal definition for computation and coordination in all applications. Without giving a fixed definition of separation criterion, Reo's structural separation of computation from coordination (i.e., component versus connector) simply means that, while this separation is always important, the distinction between the two is in the eye of the beholder: in different applications, different, or even the same people, may find it convenient to draw the line that separates computation and coordination at different places to suit their needs. For example, the stateful behavior of a `FIFO` with capacity of 1 strictly places what this entity does in the behaviour layer of BIP, as a (computation) component. In Reo, such stateful components can, of course, be regarded and used as computation as well. However, when deemed appropriate, one can use the same component (i.e., a `FIFO1` channel) in the construction of a Reo connector as well, e.g., to express the stateful, turn-taking interaction between two components, as in Figure 2.

Our data-agnostic translations allow compositional translation, because their operators distribute over composition modulo semantic equivalence. On the other hand, our data-sensitive translation scheme does not support incremental translation. It seems intuitive to translate synchronous Reo channels into BIP interaction expressions. However, the directionality inherent in the dataflows of BIP interaction expressions implies that they can compose only hierarchically, whereas the *relational* specification of dataflow constraints in Reo (which manifests itself as data constraints in constraint automata transition labels) allows more expressive composition of dataflows as relational composition of constraints. This difference restricts the set of the Reo connectors that this scheme can incrementally translate into BIP, as well as the granularity of the sub-connectors that it can translate in one increment: the data constraints on the boundary nodes of every such sub-connector must be locally resolvable into a directional dataflow expression at the level of the sub-connector, in isolation. In practice, synchronous cycles in a Reo connector must translate as a whole, which scuttles the computational benefit of translating incrementally.

In contrast with the BIP architecture model, the data-sensitive model for BIP does not include coordinating components within the connector [6, 11]. Nevertheless, it seems possible to use the formalization in [11] to extend BIP architectures of [6] with data. However, extending the current composition operator $\oplus$ to compose data-sensitive BIP architectures does not seem trivial, and we do not know what properties such an extended composition operator can preserve.

Using the ideas from Section 3, extending our reo_2 translation (Figure 3(b)) to the domain of postulated data-sensitive BIP architectures seems straight-forward. Moreover, it may be possible to extend our translations to mappings that preserve internal ports. Such extensions, together with the results from Section 4, effectively promise a property-preserving composition operator for data-sensitive BIP archi-

tectures that may also share internal ports.

References

[1] (2015): *BIP toolset*. Available at `http://www-verimag.imag.fr/BIP-Tools,93.html`.

[2] (2015): *Reo toolset*. Available at `http://reo.project.cwi.nl/reo/wiki/Tools`.

[3] Farhad Arbab (2004): *Reo: a channel-based coordination model for component composition*. Math. Structures Comput. Sci. 14(3), pp. 329–366, doi:10.1017/S0960129504004153.

[4] Farhad Arbab (2011): *Puff, The Magic Protocol*. In: *Talcott Festschrift*, Lecture Notes in Comput. Sci. 7000, Springer, pp. 169–206, doi:10.1007/978-3-642-24933-4_9.

[5] Farhad Arbab, Roberto Bruni, Dave Clarke, Ivan Lanese & Ugo Montanari (2009): *Tiles for Reo*. In: *Proc. of WADT*, Lecture Notes in Comput. Sci. 5486, Springer Berlin Heidelberg, pp. 37–55, doi:10.1007/978-3-642-03429-9_4.

[6] Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber & Joseph Sifakis (2014): *A General Framework for Architecture Composability* 8702, pp. 128–143. doi:10.1007/978-3-319-10431-7_10.

[7] Christel Baier, Joachim Klein & Sascha Klüppelholz (2014): *Synthesis of Reo Connectors for Strategies and Controllers*. Fundam. Inform. 130(1), pp. 1–20, doi:10.3233/FI-2014-980.

[8] Christel Baier, Marjan Sirjani, Farhad Arbab & Jan Rutten (2006): *Modeling component connectors in Reo by constraint automata*. Sci. Comput. Programming 61(2), pp. 75–113, doi:10.1016/j.scico.2005.10.008.

[9] Ananda Basu, Marius Bozga & Joseph Sifakis (2006): *Modeling Heterogeneous Real-time Components in BIP*. In: *Proc. of SEFM*, ACM, pp. 3–12, doi:10.1109/SEFM.2006.27.

[10] Simon Bliudze & Joseph Sifakis (2007): *The algebra of connectors: structuring interaction in BIP*. In: *Proc. of EMSOFT*, ACM SigBED, ACM, Salzburg, Austria, pp. 11–20, doi:10.1145/1289927.1289935.

[11] Simon Bliudze, Joseph Sifakis, Marius Bozga & Mohamad Jaber (2014): *Architecture Internalisation in BIP*. In: *Proc. of CBSE*, ACM, pp. 169–178, doi:10.1145/2602458.2602477.

[12] Roberto Bruni, Hernán Melgratti & Ugo Montanari (2011): *Connector Algebras, Petri Nets, and BIP*. In: *Proc. of PSI*, LNCS 7162, Springer, pp. 19–38, doi:10.1007/978-3-642-29709-0_2.

[13] M. Y. Chkouri, A. Robert, M. Bozga & J. Sifakis (2009): *Translating AADL into BIP - Application to the Verification of Real-Time Systems*. In: *Proc. of MODELS*, LNCS 5421, Springer, pp. 5–19, doi:10.1007/978-3-642-01648-6_2.

[14] K. Dokter, S.-S. T.Q. Jongmans, F. Arbab & S. Bliudze (2015): *Relating BIP and Reo*. Technical Report FM-1505, CWI. Available at `http://persistent-identifier.org/?identifier=urn:nbn:nl:ui:18-23505`.

[15] David Garlan (2014): *Software Architecture: A Travelogue*. In: *Proc. of FOSE*, ACM, pp. 29–39, doi:10.1145/2593882.2593886.

[16] Sung-Shik T. Q. Jongmans & Farhad Arbab (2012): *Overview of Thirty Semantic Formalisms for Reo*. Sci. Ann. Comp. Sci. 22(1), pp. 201–251, doi:10.7561/SACS.2012.1.201.

[17] C. Koehler & D. Clarke (2009): *Decomposing port automata*. In: *Proc. of SAC*, ACM, pp. 1369–1373, doi:10.1145/1529282.1529587.

[18] C. Krause (2009): *Integrated Structure and Semantics for Reo Connectors and Petri Nets*. In: *Proc. of ICE*, pp. 57–69, doi:10.4204/EPTCS.12.4.

[19] R. Milner (1989): *Communication and Concurrency*. Prentice-Hall, Inc.

[20] G. A. Papadopoulos & F. Arbab (2001): *Configuration And Dynamic Reconfiguration Of Components Using The Coordination Paradigm*. Future Generation Computer Systems 17(8), pp. 1023 – 1038, doi:10.1016/S0167-739X(01)00043-7.

[21] José Proença & Dave Clarke (2008): *Coordination Models Orc and Reo Compared*. Electron. Notes Theor. Comput. Sci. 194(4), pp. 57–76, doi:10.1016/j.entcs.2008.03.099.

[22] Carolyn Talcott, Marjan Sirjani & Shangping Ren (2011): *Comparing three coordination models: Reo, ARC, and PBRD*. Sci. Comput. Programming 76(1), pp. 3–22, doi:10.1016/j.scico.2009.11.006.