Intentional Automata: A Context-Dependent Model For Component Connectors
(Extended Abstract)

David Costa[14], Milad Niqui[1], and Jan Rutten[123]

[1] CWI, Amsterdam, The Netherlands.
[2] VUA, Amsterdam, The Netherlands.
[3] RUN, Nijmegen, The Netherlands.
[4] Fredhopper, Amsterdam, The Netherlands.

1 Introduction

In recent years, a promising line of research on formal compositional models for component connectors [3–6, 8, 9] has demonstrated the merits of having connectors as first class concepts, and incrementally increased the expressiveness of the interaction protocols that can be captured compositionally. Typically, in these models connectors are entitled to have their own *specification* and *abstractions*. Through composition, just like components, connectors can be *combined* and yield more sophisticated connectors. The models provide an abstract semantic domain to express interaction protocols, provide operations on the domain, and a behavioural equivalence relation of interest that identifies elements of the domain. Special elements of the domain are chosen to specify *basic* interaction protocols and correspond to the behaviour of so-called *primitive connectors*. Bruni *et al.* [8] consider four rather simple stateless primitive connectors, that essentially model synchronisation, mutual exclusion and hiding. The obtained model allows to build a wide range of coordination connectors. For instance, they are expressive enough to model the multiple-action synchronisation mechanism of CommUnity [11] which uses complex architectural connectors. Arbab and Rutten [3] and Baier *et al.* [4] consider in their models three additional primitive connectors: FIFO buffers, non-deterministic merger, and lossy channel; and define elegant compositional operational semantics for a large class of $\mathcal{R}$eo connectors. In particular, both these models are refined enough to differentiate the behaviour of non-deterministic $\mathcal{R}$eo connectors that in dataflow models, such as Kahn networks, lead to the so-called Brock-Ackerman anomalies [7]. Clarke *et al.* [9] consider an additional primitive, the *LossySync* channel, and proposed the colouring semantics as the first compositional model for context-dependent connectors. Context-dependency is a feature that extends the class of dataflow behaviour that connectors can express by including behaviour that depends on the presence or absence of *pending* I/O operations—an I/O operation present on a port. The observation of pending I/O operations permits to express in the model non-monotonic behaviour such as precedence and blocking—referred to as context-dependent behaviour. More recently Bonsangue *et al.* [6] proposed

an alternative model for context dependent-connectors based on guarded automata. The class of context-dependent behaviour captured compositionally by their models is similar to the colouring semantics, but because guarded automata can be *partial*, deficiencies identified in the colouring semantics that arise due to the totality of colouring tables not being preserved under composition, are avoided. In the context of service orchestration, Barbosa *et al.* [5] define a compositional calculus in the Bird and Meertens style where Reo-like context-dependent connectors can be modelled compositionally and show that their expressiveness permits to capture non-trivial orchestration protocols.

In this paper we contribute to that same line of research, and propose models for context-dependent connectors that permit to capture compositionally a larger class of context-dependent behaviour. Our contributions are:

1) The definition of intentional automata, and an operational semantics based on observational equivalence à la Milner, to reason compositionally about context-dependent connectors. The semantics permit to express compositionally a class of *dataflow priority* and *dataflow blocking* behaviour that includes the intended behaviour of the $LossySyncFIFO_1$ channel, and unlike all the previous models, includes context-dependent connectors constructed with multiple $FIFO_1$ channels, like the $LossySyncFIFO_2$ channel.
2) The identification of a particular class of intentional automata, called Reo automata, resulting from the axiomatisation of a Reo connector port.
3) A definition that captures the intentional automata that model context-dependent behaviour and characterises context-dependent connector.

These results have been obtained in the context of Costa's PhD research and recently detailed in his thesis [10]. In this paper due to size limitations, we skip the technical details and give an overview of the results, and for a detailed account we refer the reader to the thesis.

2 Preliminaries

We assume a set $\{A, B, C, \ldots\}$ of names that we denote by $\mathcal{N}ames$. A connector is viewed as a black box with a well-defined interface that corresponds to the collection of communication *ports*, $\Sigma \subseteq \mathcal{N}ames$, through which the connector interacts with its environment. Given a connector C with a set of ports Σ, the different ways that the environment can interact with C are given by the set of requests $\mathcal{R} = \mathcal{P}(\Sigma)$. For a *request-set* $R \in \mathcal{R}$, every port in R has a request, whereas there are no requests on ports in $\Sigma \backslash R$. For instance, consider a connector with a set of ports $\Sigma = \{A, B\}$. The set $\mathcal{P}(\{A, B\})$ contains all the request-sets the environment can perform on *Sync*. The empty request-set $\emptyset$ denotes that none of the ports of the connector receives a request from its environment. The request-set $\{A\}$ denotes that a request is present on port A, and no request is present on port B. The request-set $\{A, B\}$ denotes that a request is present on port A and another one is present on port B simultaneously. A connector C processes one *request-set* at a time, and in response produces a (possibly empty)

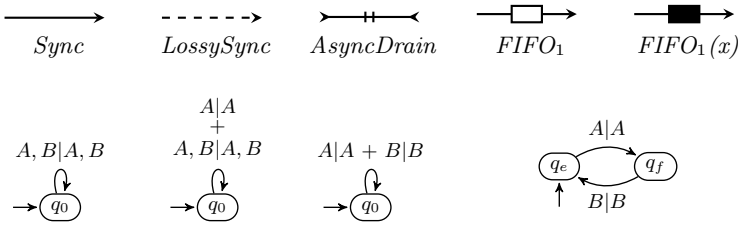

Fig. 1: Reo channel types and their intentional automata models.

firing-set $F \subseteq \Sigma$. The set of all firing-sets $\mathcal{P}(\Sigma)$ is denoted by $\mathcal{F}$. For example, the empty firing-set $\emptyset$ denotes *quiescence*—no firing at any of the ports. The firing-set $\{A, B\}$ denotes the simultaneous firing of ports A and B. A (possibly empty) request-set is related to a (possibly empty) firing-set.

We use Reo as our reference language to build connectors. Reo is a coordination language based on a calculus of channel composition to construct component connectors. Due space limitations we refer the reader to the paper by Farhad [1] for an introduction on Reo.

Figure 1 contains some common Reo channel types. *Sync* denotes a synchronous channel. Data flows through this channel if and only if it is possible to synchronously accept data on one end and pass it out through the other end. *LossySync* denotes a lossy synchronous channel. If a *take* is pending on the sink end of this channel and a *write* is requested on its source end, then the channel behaves as a synchronous channel. However, if no *take* is pending, the *write* fires, and the data is lost. Observe that this channel has *context-dependent behaviour*, as it behaves differently depending upon whether there is a *take* pending on its sink end—if it were context independent, the data could be lost regardless of whether its sink end has or does not have a *take* pending. *AsyncDrain* denotes an asynchronous drain. Data can flow into only one end of this channel at the exclusion of data flow at the other end. $FIFO_1$ denotes an empty FIFO with one buffer cell. Data can flow into the source end of this buffer, but no flow is possible at its sink end (since its buffer is empty). After data flow into the buffer cell, it becomes a full FIFO. $FIFO_1(x)$ denotes a full FIFO with one buffer cell occupied by the data value x. Data value x can flow out of the sink end of this buffer, but no flow is possible at the source end (since its buffer is full). After x flows out of the buffer, it becomes an empty FIFO.

3 Intentional Automata

To model a connector with an intentional automaton one can think of an abstract state machine, where each *state* corresponds to a possible *configuration* of the connector and the *transitions* indicate the *experiments* that take the connector from one configuration to another, not necessarily different, configuration.

Definition 1 (intentional automata). *A* non-deterministic intentional automaton *over the set of ports Σ is a system $\mathcal{A} = (Q, \Sigma, \delta, I)$, with a* set of states *Q; a* transition function *$\delta : Q \to \mathcal{P}(\mathcal{F} \times Q)^{\mathcal{R}}$ that associates for every state $q \in Q$, a function $\delta_q \in \mathcal{R} \longrightarrow \mathcal{P}(\mathcal{F} \times Q)$, where $\mathcal{R}$ are the* requests *of $\mathcal{A}$ and $\mathcal{F}$ are the* firings *of $\mathcal{A}$, and a non-empty set of initial states $I \subseteq Q$.*

The transition function δ assigns to each state $q \in Q$ the behaviour given by a function $\delta_q : \mathcal{R} \longrightarrow \mathcal{P}(\mathcal{F} \times Q)$. For each request-set $R \in \mathcal{R}$, δ_q maps R to a set of responses in $\mathcal{P}(\mathcal{F} \times Q)$. Whenever the behaviour of a state q is undefined for a particular request-set R, $\delta_q(R) = \emptyset$, and we write $\delta_q \Uparrow R$. Whenever $I = Q$ we simply write $\mathcal{A} = (Q, \Sigma, \delta)$ to denote the intentional automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$.

$\longrightarrow$ transition relation The transition function of the non-deterministic intentional automata $\delta : Q \longrightarrow \mathcal{P}(\mathcal{F} \times Q)^{\mathcal{R}}$ can be equivalently represented by a relation $\longrightarrow \subseteq Q \times \mathcal{R} \times \mathcal{F} \times Q$ defined by: $q \xrightarrow{R|F} q' \equiv (F, q') \in \delta_q(R)$.

Internal transitions Internal steps (or internal activity) of a connector account for the non-observable activity of a connector and are modelled in the automata by *internal transitions*. Internal activity in the connector takes place without involving the ports of the connector. An internal transition allows the automaton to change from a state q to state q' when no requests are present in the ports of the connector and as a result no ports of the connector are fired.

Definition 2 (internal transition). *We call a transition of type $q \xrightarrow{\emptyset|\emptyset} q'$, with $q \neq q'$, an* internal transition.

Labelled transition diagram A labelled transition diagram for an intentional automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ has its vertices labelled by the states $q \in Q$; there is a directed edge represented with an arrow labelled $R \mid F$ from the vertex labelled q to the vertex labelled q' precisely when $\delta_q(R) = (F, q')$. The initial states are distinguished by an inward-pointing arrow. If $R_1|F_1, \ldots, R_n|F_n$ label n edges from the state q to the state q' then we simply draw one arrow from q to q' labelled $R_1|F_1 + \ldots + R_n|F_n$ instead of n arrows labelled $R_1|F_1$ to $R_n|F_n$. We omit the delimiting set of brackets of the request-set R and firing-set F when labelling an edge. Irrespective of the direction of arrows, the label $R|F$ reads always from left to right: the two edges $q \xrightarrow{A,B|A} q'$ and $q' \xleftarrow{A,B|A} q$ both depict the transition $\delta_q(\{A, B\}) = (\{A\}, q')$.

Figure 1 contains the intentional automata models for some Reo channel types.

3.1 Connector Equivalence and Operations on Automata

Like in constraint automata we use bisimulation as equivalence relation [10, Def. 4.2.2, p. 78]). Unlike constraint automata we use weak-bisimulation as observational equivalence [10, Def. 4.2.7, p. 79]) and show that it is a congruence

with respect to the operations. The main reason to consider observational equivalence is because we consider internal transitions in the model. In constraint automata, all the transitions are observable and the hiding operation that removes hidden ports also eliminates what could be interpreted as internal transitions (transitions labeled with an empty set of ports).

We define three operations on intentional automata to model compositionally composite connectors [10, Section 4.3]): product, hiding, and internal transitions elimination. The product operation on intentional automata, like in constraint automata, follows the standard construction for building finite automata for intersection. It also has similarities with composition operators of process algebra, namely, the parallel composition of labelled transition systems with synchronisation over common actions and interleaving over other actions, as in TCSP [17]. The hiding operation on constraint automata performs two separate constructions: (a) it removes all information about the hidden port; (b) it performs the elimination of the transitions labelled only with the hidden port. The hiding operation on intentional automata performs also two separate constructions: (i) it removes all information about the hidden port; (ii) it prioritises the transitions in which the hidden port has a request and fires, over the transitions in which the hidden port has a request but does not fire. Construction (i) is similar to (a), whereas construction (ii) has no counterpart in constraint automata. Construction (b) is not performed by the hiding operation on intentional automata. In intentional automata, the transitions labelled only by hidden ports become internal transitions. The internal transition in intentional automata are eliminated by performing a construction similar to construction (b) on constraint automata; and both are similar to the elimination of ϵ-transitions in ordinary non-deterministic finite automata. Construction (ii) represents the main difference between the operations on the two automata models. In fact, construction (ii) uses information present only in intentional automata transitions regarding requested I/O operations, and based on this information, it prioritises the transitions. We opted for having a dedicated operation to eliminate the internal transitions and separate it from the hiding operation mainly because it makes the definition of the hiding operation easier to write, and makes more clear the parallel between the operation to eliminate internal transitions on intentional automata, with the operation to eliminate epsilon transitions from the classical theory of Finite Automata.

4 Reo Automata

Using properties that characterise how $\mathcal{R}$eo ports interact with their environment we can restrict the large class of connectors that intentional automata models to the class of models that captures $\mathcal{R}$eo connectors behaviour. In this particular class, it helps to think of states as having a structure according to the configuration of a $\mathcal{R}$eo connector. A connector *configuration* is partitioned into two parts: the *internal configuration* and the *external configuration*. An *internal configuration* is an abstract representation of the internal memory of the con-

nector. An internal configuration is denoted by an element $s \in S$, where S is the set of all internal configurations of the connector. The *external configuration* of a connector describes the status of the connector's interface and is denoted by a set $P \subseteq \Sigma$, where Σ is the set of ports of the connector. The intuition is that in a given configuration (s, P), the set P indicates the ports of the connector that have a *pending request*. We shall refer to these as *pending ports*. These are ports that have received a request in previous evaluation steps and for which the request has not been handled until now. Obviously, if a port is not in P, then this port has no pending requests.

Definition 3 (connector configuration). *Consider a connector with a set of internal configurations S and a set of ports Σ. A* configuration *is a pair (s, P) consisting of an internal configuration $s \in S$ and an external configuration $P \subseteq \Sigma$. The set of all* configurations *of the connector is given by $S \times \mathcal{P}(\Sigma)$. Given a configuration (s, P), we say p is* a pending port *or* port p has a pending request *if $p \in P$. All the configurations of a connector are initial, unless a subset of configurations $I \subseteq S \times \mathcal{P}(\Sigma)$ is defined as initial.*

Recall the transition relation of intentional automata, $\longrightarrow \subseteq Q \times \mathcal{R} \times \mathcal{F} \times Q$. We define the set of states Q as the set of configurations $S \times \mathcal{P}(\Sigma)$. A transition $(s, P) \xrightarrow{R|F} (s', P')$ models an *evaluation step* of a connector. The connector changes from configuration (s, P) to configuration (s', P') by evaluating the request-set R and producing the firing-set F.

Ports in $\mathcal{R}$eo connectors interact in a particular manner with the environment, which allows us to infer important axioms for the evaluation steps of automata models for $\mathcal{R}$eo connectors [10]:

- ♡ in a given configuration, a connector takes each pending port ($p \in P$) and requested port ($p' \in R$), and either fires it ($p \in F$ and/or $p' \in F$) or keeps it pending ($p \in P'$ and/or $p' \in P$, accordingly): $P \cup R = F \cup P'$.
- ♣ a port $p \in P$ (p is pending) implies $p \notin R$ (p cannot receive a request): $P \cap R = \emptyset$.
- ♢ a port that fires cannot become/remain pending: $F \cap P' = \emptyset$.

The $\mathcal{R}$eo connectors from the literature are expressive and constitute an interesting class to study. These connectors have an additional axiom: *None of the $\mathcal{R}$eo connector specifications distinguishes between the set of pending ports and the set of requested ports when deciding which ports to fire.* Considering an evaluation step of an automaton model for a $\mathcal{R}$eo connector, this translates to the following property: $\delta_{(s,\,P)}(R) = \delta_{(s,\,\emptyset)}(R \cup P)$. This property together with ♣ imply an additional property that characterises the transition function δ for automata models of $\mathcal{R}$eo connectors present in the literature:

$$\spadesuit \quad \delta_{(s,P)}(R) = \begin{cases} \delta_{(s,\emptyset)}(R \cup P) & \text{if } R \cap P = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

Consider a firing-set that involves multiple ports. By property ♡, the environment must perform requests on these ports before they can fire. Property ♠ says

that the firing is produced independently of the order in which the environment executes the requests on the ports that subsequently fire. Therefore we can identify the different ordering possibilities as a single one and reduce the complexity of the model. The Reo connectors enjoy properties ♡, ♠, and ♢.

Definition 4 (Reo automata). *Consider a connector C with ports Σ, internal configurations S, and a set of initial configurations $I \subseteq S \times \mathcal{P}(\Sigma)$. A* Reo automaton *for C is a non-deterministic intentional automaton $\mathcal{A}_C = (Q, \Sigma, \delta, I)$ with states $Q = S \times \mathcal{P}(\Sigma)$ and transition function $\delta : Q \to \mathcal{P}(\mathcal{F} \times Q)^{\mathcal{R}}$ that associates with every state $q = (s, P)$ a function $\delta_q : \mathcal{R} \longrightarrow \mathcal{P}(\mathcal{F} \times Q)$ such that:*

$$\heartsuit \diamondsuit \langle F, (s', P') \rangle \in \delta_{(s,P)}(R) \Longrightarrow P \cup R = F \cup P' \text{ and } F \cap P' = \emptyset$$

$$\spadesuit \quad \delta_{(s,P)}(R) = \begin{cases} \delta_{(s,\emptyset)}(R \cup P) & \text{if } R \cap P = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

We write $C = (S, \Sigma, \mathcal{A}_C)$ to denote a connector with name C, ports Σ, internal configurations S, and semantics given $\mathcal{A}_C = (Q, \Sigma, \delta, I)$ where $Q = S \times \mathcal{P}\Sigma$.

In a Reo automaton, it follows from properties ♡ ♢ and ♠ that for each internal (memory) configuration s the transitions from the state $(s, \emptyset)$ are enough to characterise the transitions of all states of the form (s, P). Furthermore, for any transition of the form $(s, \emptyset) \xrightarrow{R|F} (s, P)$ we have that $R = F \cup P$ and $F \cap P = \emptyset$.

Properties ♡ ♢ and ♠ make it possible to turn a partial intentional automaton that defines only the transition function for the states of the form $(s, \emptyset)$ into a fully specified intentional automaton. To represent a partial intentional automaton that specifies the transition function only for states of the form $(s, \emptyset)$ we define a concise tabular representation called *configuration table*.

Reo automata encode in each transition the context in which a firing occurs. With this information in the model we can define and precisely characterise context-dependent connectors like the *LossySync*.

Definition 5 (context-dependent connector). *Consider a Reo connector $C = (S, \Sigma, \mathcal{A})$. For each state (s, P) in $\mathcal{A} = (Q, \Sigma, \delta)$ we calculate the set:*

$$\Phi_{(s,P)} = \{F \mid \exists R, q \text{ s.t. } (s, P) \xrightarrow{R|F}_{\delta} q, F \neq \emptyset\}.$$

C is a context-dependent *connector if for some $s \in S$, there is a pair of different states (s, P_1), and (s, P_2) such that $\Phi_{(s,P_1)} \neq \Phi_{(s,P_2)}$.*

As an example, consider the connector $\mathit{LossySync} = (\{s\}, \{A, B\}, \mathcal{A}_{\mathit{LossySync}})$. The configuration table θ_{Sync} depicted in Figure 2, on the left, defines the transition function of the Reo automaton $\mathcal{A}_{\mathit{LossySync}}$. The corresponding labelled transition diagram is depicted on the right.
Consider the *LossySync* Reo automata model depicted in Figure 2 we have $\Phi_{(l,\emptyset)} = \{\{A\}, \{A, B\}\}$ and $\Phi_{(l,\{B\})} = \{\{A, B\}\}$. Hence, because $\Phi_{(l,\emptyset)} \neq \Phi_{(l,\{B\})}$, by definition 5 *LossySync* is a context-dependent connector.

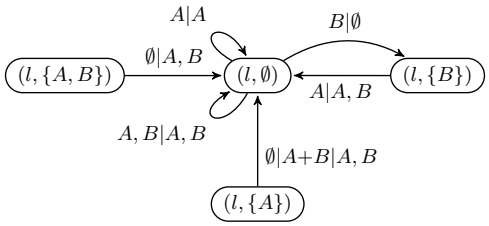

Fig. 2: The labelled transition diagram of $\mathcal{A}_{LossySync}$.

References

1. Arbab, F.: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science 14(3), 329–366 (2004)
2. Arbab, F., Chothia, T., van der Mei, R., Meng, S., Moon, Y.J., Verhoef, C.: From coordination to stochastic models of QoS. In: Field, J., Vasconcelos, V.T. (eds.) COORDINATION. LNCS, vol. 5521, pp. 268–287. Springer (2009)
3. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Wirsing, M., *et al.* (eds.) WADT. LNCS, vol. 2755, pp. 34–55. Springer (2002)
4. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by Constraint Automata. SCP 61(2), 75–113 (2006)
5. Barbosa, M.A., Barbosa, L.S.: A perspective on service orchestration. SCP 74(9), 671–687 (2009)
6. Bonsangue, M., Clarke, D., Silva, A.: Automata for context-dependent connectors. In: Field, J., Vasconcelos, V.T. (eds.) COORDINATION. LNCS, vol. 5521, pp. 184–203. Springer (2009)
7. Brock, J.D., Ackerman, W.B.: Scenarios: A model of non-determinate computation. In: Proceedings of the ICFPC. pp. 252–259. Springer-Verlag (1981)
8. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. TCS 366(1-2), 98–120 (2006)
9. Clarke, D., Costa, D., Arbab, F.: Connector Colouring I: Synchronisation and Context Dependency. SCP 66(3), 205–225 (2007)
10. Costa, D.: Formal Models For Component Connectors. Ph.D. thesis, VUA (2010). URL: http://dare.ubvu.vu.nl/handle/1871/16380.
11. Fiadeiro, J.: Categories for Software Engineering. Springer (2004)
12. Gelernter, D., Carriero, N.: Coordination languages and their significance. Commun. ACM 35(2), 97–107 (1992)
13. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation (3rd Edition). Addison-Wesley LP Co., Inc. (2006)
14. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc. (1989)
15. Park, D.: Concurrency and Automata on Infinite Sequences. In: Proceedings of the 5th GI-Conference on Theor. Comp. Science. pp. 167–183. Springer-Verlag (1981)
16. Shaw, M.: Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status. In: Lamb, D.A. (ed.) ICSE Workshop on Studies of Software Design. LNCS, vol. 1078, pp. 17–32. Springer (1993)
17. Stephen D. Brookes, Charles A. R. Hoare, and Andrew W. Roscoe. A Theory of Communicating Sequential Processes. J. ACM, 31(3):560599, 1984.