Towards a Connector Algebra

Marco Autili, Chris Chilton, Paola Inverardi, Marta Kwiatkowska, Massimo Tivoli

Towards a Connector Algebra *

Marco Autili[1], Chris Chilton[2], Paola Inverardi[1],
Marta Kwiatkowska[2], and Massimo Tivoli[1]

[1] Dipartimento di Informatica - Università degli Studi di L'Aquila, Italy
{marco.autili,paola.inverardi,massimo.tivoli}@di.univaq.it
[2] Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK
{chris.chilton,marta.kwiatkowska}@comlab.ox.ac.uk

Abstract. Interoperability of heterogeneous networked systems has yet to reach the maturity required by ubiquitous computing due to the technology-dependent nature of solutions. The CONNECT Integrated Project attempts to develop a novel network infrastructure to allow heterogeneous networked systems to freely communicate with one another by synthesising the required connectors on-the-fly. A key objective of CONNECT is to build a comprehensive theory of composable connectors, by devising an algebra for rigorously characterising complex interaction protocols in order to support automated reasoning. With this aim in mind, we formalise a high-level algebra for reasoning about protocol mismatches. Basic mismatches can be solved by suitably defined primitives, while complex mismatches can be settled by composition operators that build connectors out of simpler ones. The semantics of the algebra is given in terms of *Interface Automata*, and an example in the domain of instant messaging is used to illustrate how the algebra can characterise the interaction behaviour of a connector for mediating protocols.

1 Introduction

Ubiquitous computing is an emerging paradigm that is rapidly changing the way we use technology to perform everyday tasks. The widespread availability of digital systems, together with the introduction of new communication infrastructures, make it possible to run and interact with software systems on a variety of networked devices. However, computing and networking technologies have yet to reach the maturity required by ubiquitous computing since technology-dependent limitations reduce the effectiveness of integrating and composing heterogeneous networked systems.

The CONNECT Integrated Project[3] attempts to develop a novel networking infrastructure to allow heterogeneous networked systems to freely communicate with one another. This would be achieved by the synthesis of emergent connectors on-the-fly. Towards this aim, a key objective of CONNECT is to build

* This work is partly supported by the CONNECT European Project No 231167, and EPSRC project EP/D076625/2.

[3] http://connect-forever.eu/.

a comprehensive theory of composable connectors, by devising an algebra that can model complex interaction behaviours with respect to both functional and non-functional properties. The algebra will serve as a baseline for automated reasoning and learning about system interaction behaviours, in addition to automated synthesis, matching, refinement, composition, evolution, and (possibly partial) re-use of connectors. This also concerns finding an adequate formalism to express and quantify, for each connector, the desired Quality of Service levels for end-to-end properties of the networked systems.

The comprehensive characterisation of such a connector algebra is our long-term goal. In this paper, as a starting point for developing such an algebra, we focus only on the functional behaviour of connectors that act as *protocol mediators*. Consequently, our algebra will characterise the behavioural mismatches that occur during interactions among heterogeneous networked systems. Quantitative aspects of networked systems and connectors are not considered hereafter; that is left as future work.

As discussed in [14,19] (and references therein), a possible approach to protocol mediation involves the categorisation of recurring protocol mismatches that must be solved by means of *mediator patterns*. For each type of mismatch, a pattern can be defined as a solution to the interaction incompatibility. Clearly, a catalogue of such problems and their related solutions would not solve all possible mismatches, but combining multiple sub-solutions should facilitate their solution.

Inspired by the set of basic mediator patterns described in [19], this paper presents a high-level algebra that reasons about protocol mismatches. Solutions to basic mismatches are modelled as primitives of the algebra, while complex mismatches can be solved by combining suitable primitives in a variety of ways. The semantics of the algebra is given in terms of *Interface Automata* (IA) of de Alfaro and Henzinger [11]. Thus, composition of terms in the algebra reduces to composition of the underlying IA. Some of these compositions are already defined on IA, but we will also present specific compositions that we conjecture are necessary for a meaningful connector algebra.

Our choice of using IA for the semantics of the algebra is heavily influenced by a previous survey of connector notations we conducted as part of Connect [1]. In that report we surveyed a number of formalisms against eight dimensions deemed to be of particular interest to the project. These were compositionality, incrementality, scalability, compositional reasoning, reusability, evolution, ability to express and reason about non-functional properties, and the existence of a specialised notation supported by automated tools for architectural analysis.

The choice of formalisms was decided to give a thorough coverage of the space of connectors. To give an indication: some formalisms were control-oriented [2], while others were data-flow based [7]; some supported hierarchy [9], whereas others provided mobility [18]. In the spirit of Connect, we also surveyed quantitative extensions of [7]; these included an extension involving discrete probabilities with non-determinism [6], a stochastic extension [4], and an extension with Quality of Service attributes [5]. Beyond the surveyed approaches, we also

investigated a number of other formalisms, but space limitations prevents us from elaborating upon them here.

Summarising the results of our survey and investigations led us to the opinion that IA are the most suitable formalism for modelling connectors. This was based on the fact that IA can be extended to support reasoning on non-functional properties, together with compositionality results implying reuse and evolution of connectors. IA are by no means complete in satisfying the properties we require of a connector algebra, but they do give us a good starting point and seem extensible enough to gain a good coverage of the dimensions of interest.

The remainder of this paper is organised as follows. Section 2 recalls background notions concerning IA, and offers justification for choosing these devices. Section 3 describes a scenario that we will use to demonstrate the effectiveness of our algebra, with models given in terms of IA. Following on, Section 4 introduces the algebra in a formal way and concludes by relating the algebra to the case study presented in Section 3. Finally, Section 5 summarises our work and discusses possible future research directions.

2 Semantics for Connectors

As clarified in Section 1, it is our intention to ascribe semantics to our algebra in terms of *Interface Automata* (IA) [11]. As we shall see later on, IA do not give us all of the desired functionality and properties that we require to interpret our algebra, but they do give us a good starting point.

IA may be seen as finite state machines whose actions are partitioned into input and output sets. At the syntactic level this classification has no significance, but examination of the semantics reveals a notion of communication well suited to components interacting in a heterogeneous environment.

Definition 1. *An Interface Automaton is a tuple* $(V, v_0, \mathcal{A}_I, \mathcal{A}_O, \rightarrow)$*, where:*

- V *is a set of states*
- $v_0 \in V$ *is the designated initial state*
- $\mathcal{A} = \mathcal{A}_I \cup \mathcal{A}_O$ *is the set of actions.* $\mathcal{A}_I$ *and* $\mathcal{A}_O$ *are disjoint sets referred to as the input actions and output actions respectively.*
- $\rightarrow : V \times \mathcal{A} \rightharpoonup V$ *is the transition (partial) function.*

For brevity, we write $v \xrightarrow{a} v'$ iff $\rightarrow (v, a) = v'$. The partial transition function ensures that the automaton is loosely deterministic (there is at most one successor for each state-action pair). This contrasts with the I/O automaton model of Lynch and Tuttle [16], where every state must be fully input-enabled.

The automaton behaves in a manner similar to that for finite state machines; it starts in the initial state and evolves over time according to its transition function. However, special consideration must be given to the transition types. Following the finite state case, a transition labelled by an input action may only be picked when the environment is offering that action. Output actions are non-blocking, so may be taken at any time if they are enabled in the current state.

Since outputs are non-blocking, it follows that the environment must be willing to accept any action it is offered.

Until now, IA have appeared as marginally generalised I/O automata. Their overarching power becomes apparent when we consider the composition of multiple IA, and observe how they interact with the environment.

From hereon let $\mathcal{C} = (V^C, v_0^C, \mathcal{A}_I^C, \mathcal{A}_O^C, \rightarrow_C)$ and $\mathcal{D} = (V^D, v_0^D, \mathcal{A}_I^D, \mathcal{A}_O^D, \rightarrow_D)$ be two IA. $\mathcal{C}$ and $\mathcal{D}$ may only be composed if their action sets are compatible with each other. Consequently, not every pair of IA can be composed.

Definition 2. *IA $\mathcal{C}$ and $\mathcal{D}$ are said to be* composable *just if $\mathcal{A}_O^C \cap \mathcal{A}_O^D = \emptyset$. Outputs must be disjoint to prevent synchronisation.*

To define the (parallel) composition on composable IA we begin by constructing the product of the automata. Unfortunately, this can introduce a number of illegal states, whereby one of the automata is willing to offer an output action in the common alphabet, but the second is not able to offer the corresponding input action. This is a consequence of not requiring IA to be fully input-enabled.

Definition 3. *The product of $\mathcal{C}$ and $\mathcal{D}$ is an interface automaton $\mathcal{C} \otimes \mathcal{D} = (V^C \times V^D, (v_0^C, v_0^D), \mathcal{A}_I, \mathcal{A}_O, \rightarrow)$, where:*

- $\mathcal{A}_I = (\mathcal{A}_I^C \cup \mathcal{A}_I^D) \setminus ((\mathcal{A}_I^C \cap \mathcal{A}_O^D) \cup (\mathcal{A}_O^C \cap \mathcal{A}_I^D))$
- $\mathcal{A}_O = \mathcal{A}_O^C \cup \mathcal{A}_O^D$
- *The transition function is given by*

$$(s,t) \xrightarrow{a} (s',t') \Leftrightarrow \begin{cases} s \xrightarrow{a}_C s' \wedge t \xrightarrow{a}_D t' & \text{if } a \in \mathcal{A}^C \cap \mathcal{A}^D \\ s \xrightarrow{a}_C s' \wedge t = t' & \text{if } a \in \mathcal{A}^C \setminus \mathcal{A}^D \\ t \xrightarrow{a}_D t' \wedge s = s' & \text{if } a \in \mathcal{A}^D \setminus \mathcal{A}^C. \end{cases}$$

We must now identify the illegal states in the product $\mathcal{C} \otimes \mathcal{D}$, denoted by $\mathit{Illegal}(\mathcal{C}, \mathcal{D})$. The kernel of the illegal states is taken to be the set of states (p, q) for which there is some $a \in (\mathcal{A}_I^C \cap \mathcal{A}_O^D) \cup (\mathcal{A}_O^C \cap \mathcal{A}_I^D)$ such that one of p and q can make an a-labelled output transition, but the other cannot match it with the corresponding input transition. The illegal set is then taken to be all those states in the kernel, plus those that can reach a state in the kernel by a sequence of transitions labelled by output actions.

Definition 4. *The* composition *of two composable IA $\mathcal{C}$ and $\mathcal{D}$, written as $\mathcal{C} \,||\, \mathcal{D}$, is defined to be $\mathcal{C} \otimes \mathcal{D}$ after pruning all states in $\mathit{Illegal}(\mathcal{C}, \mathcal{D})$, providing the initial state (v_0^C, v_0^D) is contained within the remaining automaton. Otherwise, the composition is undefined.*

In the words of de Alfaro and Henzinger [11], the pruning yields an optimistic notion of composition. The pruning is equivalent to making the assumption that the environment will not issue inputs that can lead to illegal states. Consequently, composed IA place an assumption on the environment about what inputs will be issued and when. This is in stark contrast to I/O automata where the composition must handle any input the environment offers.

From a system designer's point-of-view, we are interested in modelling concrete connectors by means of a term in our algebra. We could certainly develop an algebra that addresses this sole goal, however we are looking for something considerably more high-level. To that extent, and conflating the desiderata from Section 1, we believe that our algebra should also support abstractions of connectors, or more precisely specifications.

IA have an explicit correspondence with specification theories, which is why we are using them to express the semantics of our algebra. In [11], de Alfaro and Henzinger define a refinement relation $\sqsubseteq$[4] on IA in terms of alternating simulation[5] [3]. Consequently, we have a test for when a connector can be safely substituted with another. Since refinement on IA is a congruence with respect to composition (see next theorem), substitution can be performed compositionally.

Theorem 1. *For suitable restrictions on the composability of $\mathcal{C}$, $\mathcal{C}'$ and $\mathcal{D}$, if $\mathcal{C} \parallel \mathcal{D}$ is defined and $\mathcal{C} \sqsubseteq \mathcal{C}'$, then $\mathcal{C}' \parallel \mathcal{D}$ is defined and $\mathcal{C} \parallel \mathcal{D} \sqsubseteq \mathcal{C}' \parallel \mathcal{D}$.*

Besides parallel composition, a conjunctive operator $\wedge$ can be defined on IA. $\mathcal{C} \wedge \mathcal{D}$ yields the least specified interface automaton that refines both $\mathcal{C}$ and $\mathcal{D}$. This is of direct use in defining a connector that must satisfy multiple specifications. More verbosely, this means we can build up specifications in a distributed and compositional manner. This supports the *separation of concerns* principle, and again allows for compositional development.

IA can be synthesised in a specification theory by means of a quotienting operator $\backslash$, as described in [8]. Given IA $\mathcal{D}$ and $\mathcal{E}$, $\mathcal{E} \backslash \mathcal{D}$ yields the least specified interface automaton $\mathcal{C}$ such that $\mathcal{E} \sqsubseteq \mathcal{C} \parallel \mathcal{D}$. Thus, given a specification of what a connector should do, together with an implementation of a connector that implements part of that behaviour, we can synthesise a connector that when composed with the partial connector fulfils the desired behaviour. The advantages of such a feature are self-evident.

Applicability to notions of specification was a key justification for adopting IA, yet another essential reason relates to their extensibility. We would eventually like to develop an algebra that can handle a number of quantitative extensions, such as time and probability. A timed-extension of I/O automata has already been developed [10], and the model completely supports all of the specification constructs and relations we have mentioned. Furthermore, there is a probabilistic extension of I/O automata [20], although this is not given in terms of a specification theory. Nevertheless, recent work has augmented interactive Markov chains with specification notions [21], with a bridge between these and probabilistic I/O automata seeming highly plausible.

[4] $P \sqsubseteq Q$ if, and only if, P is refined by Q. Refinement tends to be one of the most prevalent relations in any specification theory.

[5] The simulation must be alternating as the inputs and outputs of a refining automaton must be related contravariantly to those of the original.

Since IA and I/O automata are closely related to each other, we would like to combine the aforementioned quantitative extensions with the optimistic composition techniques of IA. Accordingly, IA appear to support many of the features we would like of a connector algebra. Section 4 will demonstrate just how effective they are when we assign semantics to the algebra.

3 Case Study

To illustrate in Section 4 the efficacy with which our algebra can model and reason about protocol mediators, we present a simple yet challenging example of universal instant messaging inspired by [14].

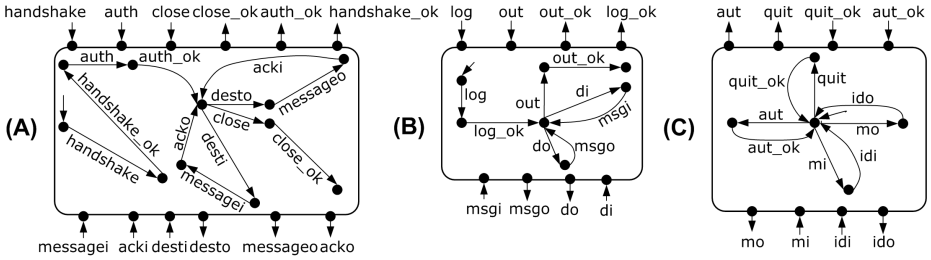

Fig. 1. (A) MSN messaging service. (B) XMPP messaging service. (C) CFring client.

Fring[6] is an instant messaging program that allows one to exchange text messages between a predefined set of heterogeneous messaging services. At present, the service supports connection to the *MSN Messenger* and *XMPP Messenger* services, amongst many others, with the collection of supported services being static. This contrasts with the evolving world of Connect, where messaging services to be bridged should not be known *a priori*. We therefore propose a generalisation of Fring, let it be called *CFring*, where connectors between unknown messaging services are generated on-the-fly. This generalisation will allow us to express the full power of the algebra.

As the interface automaton in Figure 1.C shows, the CFring service provides only core functionalities for "abstract" authentication and message exchange. In particular, when receiving (resp., sending) a message *mi* (resp., *mo*), CFring expects to receive (resp., send) the identity *idi* (resp., *ido*) of the sender (resp., receiver) as well.

The behavioural models of MSN and XMPP, which are unknown to CFring, are expressed as IA in Figures 1.A and 1.B. In contrast to the behavioural model

[6] http://www.fring.com.

for CFring, both MSN and XMPP when receiving (resp., sending) a message expect to have provided (resp., provide) the identity of the sender (resp., receiver) first. Furthermore, unlike the others, MSN expects to receive (resp., send) an acknowledgement for the message sending (resp., receiving).

It is obvious that the MSN and XMPP services should both be able to interoperate with CFring, since they amount to supporting authentication and then message exchange. This requires "specialising" the CFring communication protocol in order to mediate the communication between the other messaging services. Note that this is far from trivial, especially if one wishes to rigorously characterise the achieved interoperability (e.g., for supporting automated reasoning, detecting possible mismatches, etc.). Nevertheless, in Section 4 we realise such a connector that mediates the communication between CFring and MSN/XMPP as a term of our algebra.

4 Towards a Connector Algebra: primitives and operators

Section 1 briefly mentioned a number of existing connector formalisms that we had surveyed in [1]. The formalisms vary quite considerably; some support hierarchical development, whereas others have resolute granularity. Some support mobility, while others assume a fairly static environment. In essence, the formalisms have niche environments where they work well, while outside their haven of assumptions the quality of modelling is often variable.

In CONNECT, we are interested in generating connectors on-the-fly to bridge communication inconsistencies at both the application and middleware layers. For the purposes of this paper, we are concerned with exchanging structured messages between components, rather than worrying about, say, data transfer at the transport level.

In [19], the authors attempt to characterise mismatches between functionally equivalent yet behaviourally different protocols that wish to communicate. For each type of communication discrepancy they provide a mediating connector that can handle and resolve the mismatch. This is a high-level approach to analysing and addressing interoperability issues.

Following this insight, it is our intention to develop a high-level algebra for reasoning about mismatches. We shall model each basic mismatch solution as a primitive of our algebra, with semantics given in terms of IA. Complex mismatches can be decomposed into combinations of basic ones, and so an algebraic connector for a complex mismatch can be obtained by composing primitive connectors. For most cases, composition of terms in the algebra will reduce to composition of the underlying IA as described in Section 2. However, as we shall see, we will also require our own specific operators on connectors.

Components wishing to communicate with each other are modelled by arbitrary IA, which we assume have disjoint action sets[7]. We will treat the action sets associated with IA as sets of message ports that can send (resp., receive) a

[7] Under the aegis of CONNECT, equivalence of actions is assumed to be specified in an ontology. This allows us to assume disjoint component actions.

signal depending on whether they are an output (resp., input) port. Hence, at this stage we do not allow for the exchange of data over a domain.

From hereon let $\mathcal{A}$ be a global set of message ports. The primitives of the connector algebra $\mathcal{AP}(\mathcal{A})$ corresponding to the mismatches in [19] are described below.

1. **Extra send.** This first mismatch considers a component that generates a redundant message a. Such a mismatch may be resolved by introducing a consumer that swallows the superfluous message. We model this by a parameterised primitive $Cons(a)$.
2. **Missing send.** This mismatch describes the case in which a component expects a message a that is not sent by another component. A mismatch of this type may be resolved by introducing a producer that generates the required message. This may be modelled by a parameterised primitive $Prod(a)$.
3. **Signature mismatch.** There are occasions when a message to be exchanged between two components is functionally compatible yet syntactically inconsistent. In the case of CONNECT, the functional equivalence of the messages a and b is assumed to be specified in an ontology. Such a mismatch may be resolved by means of a translating primitive $Trans(a, b)$ that accepts message a as input and produces message b as output.
4. **Split message mismatch.** A component may expect to receive a message a as a sequence of fragments of a. If message a can be decomposed into $a_1, \ldots, a_n$, then the mismatch may be resolved with a primitive $Split(a, [a_1, \ldots, a_n])$ which accepts message a as input and offers $a_1, \ldots, a_n$ as output in that order.
5. **Merge message mismatch.** Similar to the previous case, some components expect to receive a single message a in place of a fragmented version of a. If messages $a_1, \ldots, a_n$ can be composed into a, then the mismatch may be resolved with a primitive $Merge([a_1, \ldots, a_n], a)$ which accepts messages $a_1, \ldots, a_n$ as input in that order, and generates a as output.
6. **Ordering mismatch.** A component can expect to receive messages in an order different from the order used by the sending component. The mismatch can be resolved by introducing an ordering primitive $Order([a_1, \ldots, a_n], \pi, [a'_1, \ldots, a'_n])$, where π is a permutation of $\{1, \ldots, n\}$. The primitive accepts inputs from one component in the order $a_1, \ldots, a_n$, and produces outputs for the other in the order $a'_{\pi(1)}, \ldots, a'_{\pi(n)}$. Note that port a_i is related to port a'_i.

Besides the mismatch primitives above, a further primitive is required to force the algebra to work in a sensible way. The primitive does not perform any interactions, so is fittingly called *NoOp*. Equipped with all of the basic primitives, a term s of the algebra is given by the following grammar:

$$s ::= s \odot s \mid s + s \mid s \wedge s \mid s \setminus s \mid s^{\perp} \mid (s) \mid p$$

$$p ::= NoOp \mid Cons(a) \mid Prod(a) \mid Trans(a, b) \mid Split(a, [a_1, \ldots, a_n]) \mid$$

$$Merge([a_1, \ldots, a_n], a) \mid Order([a_1, \ldots, a_n], \pi, [a'_1, \ldots, a'_n])$$

where $a, a_i, a'_i, b \in \mathcal{A}$ and π is a permutation of $\{1, \ldots, n\}$. The symbols $\odot$, $+$, $\wedge$, and $\setminus$ are binary operators called *plugging*, *alternation*, *conjunction* and *quotienting* respectively, and $^{\perp}$ is a unary operator called *inversion*.

The semantics of the algebra $\mathcal{AP}(\mathcal{A})$ is given in terms of a function $[\![\cdot]\!] : \mathcal{AP}(A) \to \mathcal{IA} \cup \{Err\}$, where $\mathcal{IA}$ is the universal set of IA and Err represents the undefined IA. For any term s, the denotation $[\![s]\!]$ is defined inductively.

First and foremost, if s is a primitive, then $[\![s]\!]$ is the corresponding interface automaton defined in Fig. 2, providing the parameters are well-defined (otherwise the semantics of the primitive is taken to be Err). The parameters of each primitive are single or lists of uninterpreted message ports of $\mathcal{A}$. Lists must have finite length, and message ports in the parameters of a primitive must be pairwise disjoint. In the case of $Merge$ and $Split$, we require that a and $a_1 \ldots a_n$ are equated in the ontology. Furthermore, for the case of the $Order$ primitive, we require that both lists have the same length.

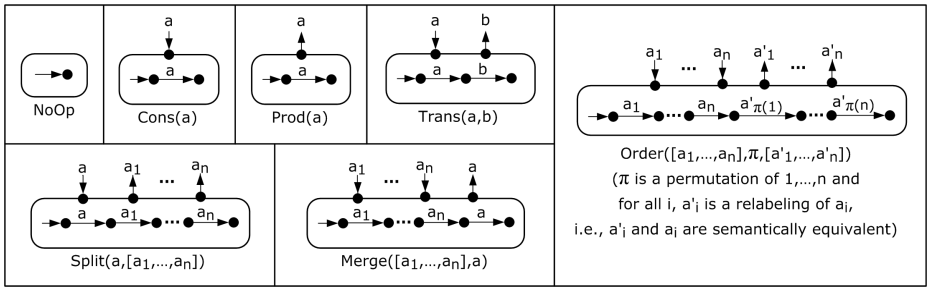

Fig. 2. Semantics of the primitives.

If s is a compound term (i.e., consists of operators), then $[\![s]\!]$ is given by the mappings below. However, an informal description is in order first. An operator on terms of the algebra induces a behaviour on the behaviours of the operands. The operator $\odot$ connects terms of the algebra on common message ports; this is equivalent to plugging the corresponding IA into each other, or synchronising them. On the other hand, the operator $+$ behaves like alternation in regular expressions; a connector defined in terms of $+$ behaves like one of its operands. The operators $\wedge$ and $\setminus$ were both defined in Section 2. Finally, $^{\perp}$ acts like the inverse of its operand by interchanging inputs and outputs.

- $s = t \odot u$. If either of $[\![t]\!]$ or $[\![u]\!]$ is equal to Err, then $[\![s]\!] = Err$. Alternatively, if $[\![t]\!]$ and $[\![u]\!]$ are not composable or $[\![t]\!] \mid\mid [\![u]\!]$ is not defined, then $[\![s]\!] = Err$. Otherwise, $[\![s]\!] = [\![t]\!] \mid\mid [\![u]\!]$.

- $s = t + u$. If either of $[\![t]\!]$ or $[\![u]\!]$ is equal to Err, then $[\![s]\!] = Err$. Otherwise, $[\![t]\!]$ and $[\![u]\!]$ are IA $\mathcal{C}$ and $\mathcal{D}$. If $\mathcal{A}_I^C \cup \mathcal{A}_I^D$ and $\mathcal{A}_O^C \cup \mathcal{A}_O^D$ are not disjoint, then the alternation is not defined and $[\![s]\!] = Err$. For the case when the action sets do agree, $[\![s]\!] = Determinise(\mathcal{C} + \mathcal{D})$, where $\mathcal{C} + \mathcal{D}$ is defined to be the IA $(V, v_0, \mathcal{A}_I^C \cup \mathcal{A}_I^D, \mathcal{A}_O^C \cup \mathcal{A}_O^D, \rightarrow)$ such that:
  - $V = V^C \dot{\cup} V^D \dot{\cup} \{v_0\}$ i.e., V^C, V^D and $\{v_0\}$ are pairwise disjoint
  - $\rightarrow = \rightarrow_C \cup \rightarrow_D \cup \{(v_0, a, v) : v_0^C \xrightarrow{a}_C v\} \cup \{(v_0, a, v) : v_0^D \xrightarrow{a}_D v\}$.

  $Determinise(\mathcal{C} + \mathcal{D})$ is the deterministic IA equivalent to the possibly non-deterministic IA $\mathcal{C} + \mathcal{D}$. Thus, $Determinise$ is a function on IA which implements a suitable variant of the algorithm described in [13], that determinises a finite state automaton.

- $s = t \wedge u$. If either of $[\![t]\!]$ or $[\![u]\!]$ is equal to Err, then $[\![s]\!] = Err$. Otherwise, $[\![t]\!]$ and $[\![u]\!]$ are IA $\mathcal{C}$ and $\mathcal{D}$. If $\mathcal{A}_I^C \cup \mathcal{A}_I^D$ and $\mathcal{A}_O^C \cup \mathcal{A}_O^D$ are not disjoint, then the conjunction does not exist and $[\![s]\!] = Err$. For the case when the action sets do agree, $[\![s]\!]$ is an IA $(V^C \times V^D, (v_0^C, v_0^D), \mathcal{A}_I^C \cup \mathcal{A}_I^D, \mathcal{A}_O^C \cap \mathcal{A}_O^D, \rightarrow)$, where $\rightarrow$ is the smallest relation satisfying the following rules:
  1. If $p \xrightarrow{a}_C p'$ and $q \xrightarrow{a}_D q'$ with $a \in \mathcal{A}_O^C \cap \mathcal{A}_O^D$, then $(p, q) \xrightarrow{a} (p', q')$
  2. If $p \xrightarrow{a}_C p'$ with $a \in \mathcal{A}_I^C \setminus \mathcal{A}_I^D$, then $(p, q) \xrightarrow{a} (p', q)$
  3. If $q \xrightarrow{a}_D q'$ with $a \in \mathcal{A}_I^D \setminus \mathcal{A}_I^C$, then $(p, q) \xrightarrow{a} (p, q')$
  4. For $a \in \mathcal{A}_I^C \cap \mathcal{A}_I^D$:
     (a) If $p \xrightarrow{a}_C p'$ and $q \not\xrightarrow{a}_D$, then $(p, q) \xrightarrow{a} (p', q)$
     (b) If $p \not\xrightarrow{a}_C$ and $q \xrightarrow{a}_D q'$, then $(p, q) \xrightarrow{a} (p, q')$
     (c) If $p \xrightarrow{a}_C p'$ and $q \xrightarrow{a}_D q'$, then $(p, q) \xrightarrow{a} (p', q')$.

- $s = t \setminus u$. Based on [8], it follows that quotienting is a derived operator of the algebra. Thus, $[\![s]\!] = [\![(t^\perp \odot u)^\perp]\!]$.

- $s = t^\perp$. If $[\![t]\!] = Err$, then $[\![s]\!] = Err$. Otherwise, $[\![s]\!]$ is equal to $[\![t]\!]$ with the input and output sets exchanged in the signature of $[\![t]\!]$.

- $s = (t)$. Simply $[\![s]\!] = [\![t]\!]$.

The operators $\odot$, $+$, $\wedge$, $\setminus$ and $^\perp$ satisfy a number of axioms, as we briefly elaborate below.

1. Plugging $\odot$ is commutative and associative, but is not idempotent. It has an identity element $NoOp$, so $(\mathcal{AP}(\mathcal{A}), \odot, NoOp)$ is a commutative monoid (i.e., an abelian semigroup with an identity). Plugging does not distribute over $+$, $\wedge$ nor $\setminus$.
2. Alternation $+$ is commutative, associative, and idempotent. The identity of $+$ is also $NoOp$, so $(\mathcal{AP}(\mathcal{A}), +, NoOp)$ is a commutative monoid. Alternation does not distribute over $\odot$ nor $\setminus$, however it does distribute over $\wedge$.

3. Conjunction $\wedge$ is commutative, associative, and idempotent. The operator does not have an identity element in the algebra, and does not distribute over $\odot$ nor $\backslash$, but it does distribute over $+$.
4. Quotienting $\backslash$ is not associative, commutative, nor idempotent. *NoOp* is a right-identity element for the operator. Quotienting does not left or right distribute over $\odot$, $+$, nor $\wedge$.
5. Inversion $^{\perp}$ distributes over $+$, but not $\odot$, $\wedge$ nor $\backslash$. Double inversion of a term is an identity function on that term.

As we remarked in Section 2, a desirable property of a connector algebra is its ability to support notions of specification. Consequently, there should be a concept of refinement on terms, and indeed our algebra does support this.

Definition 5. *Let s and t be terms of the algebra, and let $\sqsubseteq$ be the alternating simulation refinement relation defined on IA $[\![s]\!]$ and $[\![t]\!]$. Term t refines term s, written as $s \trianglelefteq t$, iff the denotation of t refines the denotation of s at the semantic level or $[\![s]\!] = Err$. Formally, $s \trianglelefteq t \Leftrightarrow [\![s]\!] \sqsubseteq [\![t]\!] \vee [\![s]\!] = Err$.*

Establishing refinement on terms allows us to define equivalence. Semantic equality of terms is too strong for equivalence of connector behaviours, so we choose to express it in terms of the weaker refinement relation.

Definition 6. *Term s is said to be equivalent to term t, written as $s \equiv t$, if, and only if, $s \trianglelefteq t$ and $t \trianglelefteq s$.*

Our choice of equivalence seems most natural for connector behaviours, as it allows for seamless substitutivity. However, it is unfortunate that the equivalence is expressed in terms of the underlying semantics, rather than the syntax of the terms. Nevertheless, this correspondence with the IA semantics means that the algebra is trivially both sound and complete, even after equating all incompatible and undefined terms with Err.

Theorem 2. *Let $\hat{=}$ denote the equivalence of IA (i.e. mutual refinement). For any terms s and t it holds that $s \equiv t \Leftrightarrow ([\![s]\!] \hat{=} [\![t]\!]) \vee ([\![s]\!] = Err = [\![t]\!])$.*

Having defined equivalence and established that the algebra is sound and complete, it is an easy consequence that our axiomatisation is correct. The reason for $\odot$ failing to be idempotent is closely related to the reason that $[\![s]\!] \,||\, [\![s]\!]$ is not defined in general, because of restrictions on composability.

The formal definition of the semantics, as well as the example of idempotence failing, has a notable consequence for the algebra. If s and t are well-formed terms whose semantics are not equal to Err, then it is not the case that the semantics of $s \odot t$, $s + t$, $s \wedge t$ and $s \setminus t$ are not equal to Err, because of the restrictions imposed by these operators. This seems undesirable, but it is a direct consequence of IA.

This shortcoming might seem unpalatable at first, but we do not believe it to be a problem; in fact, it is an advantage. In the context of Connect, we are concerned with generating connectors in a compositional manner. If we take two

connectors, each of which is expressed by a term in the algebra, we can combine the two terms and observe if the outcome is equal to Err. If it is, then it follows that the two connectors do not work with each other. Thus the algebra allows for compositional reasoning.

Another requirement of CONNECT is the ability of our algebra to serve as a baseline for automated connector synthesis, as stated below. This is closely related to the quotienting operator defined in [8], but requires suitable modification to be applicable to the algebra.

> CONNECTOR SYNTHESIS
> *Instance:* Components $\mathcal{E}$ and $\mathcal{F}$ represented by IA.
> *Problem:* Find a term $x \in \mathcal{AP}(\mathcal{A})$ such that $inv(\mathcal{E}) \sqsubseteq \mathcal{F} \mid\mid [\![x]\!]$ and $inv(\mathcal{F}) \sqsubseteq \mathcal{E} \mid\mid [\![x]\!]$, where inv inverts input and output actions.

The connector synthesis problem aims to find a connector x expressible in our algebra that can mediate interoperability incompatibilities between components represented by arbitrary IA. We require that (i) every interaction exhibited by $inv(\mathcal{F})$ is allowed by $[\![x]\!] \mid\mid \mathcal{E}$, and (ii) every interaction exhibited by $inv(\mathcal{E})$ is permitted by $[\![x]\!] \mid\mid \mathcal{F}$. This allows us to formally characterise *interoperability* between components in our algebra.

CFring example. A connector for the scenario described in Section 3 may be expressed in terms of the algebra $\mathcal{AP}(\mathcal{A})$ as the following compound term:

```
((Trans(aut,handshake) ⊙ Cons(handshake_ok) ⊙ Prod(auth) ⊙ Trans(auth_ok,aut_ok)) ⊙
(Trans(quit,close) ⊙ Trans(close_ok,quit_ok)) ⊙
(Order([mo,ido],(2,1),[mo',ido']) ⊙ Trans(ido',desti) ⊙ Trans(mo',messagei) ⊙ Cons(acko))⊙
(Order([desto,messageo],(2,1),[desto',messageo']) ⊙ Trans(messageo',mi) ⊙
Trans(desto',idi) ⊙ Prod(acki)))
+
((Trans(aut,log) ⊙ Trans(log_ok,aut_ok)) ⊙
(Trans(quit,out) ⊙ Trans(out_ok,quit_ok)) ⊙
(Order([mo,ido],(2,1),[mo',ido']) ⊙ Trans(ido',di) ⊙ Trans(mo',msgi)) ⊙
(Order([do,msgo],(2,1),[do',msgo']) ⊙ Trans(msgo',mi) ⊙ Trans(do',idi)))
```

This expression seems quite complex, but it is worthwhile noticing how it can easily be decomposed into distinct portions corresponding to the original protocols. There is close correspondence between the four branches of the CFring protocol shown in Figure 1.C, and the paths of the MSN and XMPP protocols shown in Figures 1.A and 1.B, respectively. Each of these branches neatly map onto a sub-term of the connector expression above. This suggests that connector terms can be defined by analysis of the corresponding protocols' transition systems. Unfortunately, the connector only allows the sending and receiving of a single message, but we shall elaborate on this observation further in Section 5.

Relating the connector synthesis problem to our example, we built our connector by constructing two sub-connectors x' and x''. The term x' was used

to mediate MSN and CFring. Note how $inv(MSN) \sqsubseteq [\![x']\!] \mid\mid CFring$ and $inv(CFring) \sqsubseteq [\![x']\!] \mid\mid MSN$. Analogously, x'' mediates XMPP and CFring. We combined x' and x'' by means of a suitable composition operator (i.e., $+$), thus obtaining x. Automating this kind of reasoning represents a specific area that we wish to explore, in order to develop a comprehensive theory of composable connectors in CONNECT.

5 Concluding Remarks

In this paper, we formalised an initial high-level connector algebra for reasoning about protocol mismatches. Solutions to basic mismatches are modelled as primitive terms and complex mismatches are solved by combining primitives of the algebra by means of different composition operators. The semantics of the terms and operations on them are given by IA, which is a suitable candidate for expressing the behaviour of connectors that reside within a highly heterogeneous environment, as we shall remark later.

Our formulation sets the scene for a yet-to-come comprehensive algebra facilitating the rigorous characterisation of complex interaction behaviours. Such an algebra should allow reasoning with respect to both functional and non-functional properties, in addition to supporting automated reasoning and learning about system interaction behaviour, as well as automated synthesis, matching, refinement, composition, evolution, and (possibly partial) re-use of connectors.

The case study highlights a shortcoming of our current algebra, in that we cannot construct a connector that exhibits looping behaviours. The form of our algebra dictates that for any term whose semantics are equivalent to an IA (as opposed to Err), the structure of the automaton is a directed graph in which every state is visited at most once. Such a restriction on the behaviour of the connector is unduly restrictive. This is evident from our case study, where the connector only supports the sending and receiving of a single message. If the number of messages to be sent and received is known in advance, then we can build a connector whose size is related polynomially to the number of messages to be transmitted. This, however, is inadequate.

In a future version of the algebra, the restriction on looping would need to be lifted. We already have an idea of how this could be done, by introducing looping equivalents of the primitives. It also seems likely that we would need a fix-point operator to encode complex looping behaviours in the algebra, beyond those at the primitive level. It would be interesting to see whether looping operators make it easier to model connectors in our algebra or not. Naturally, we would hope so.

This shortcoming of the algebra is not to say that IA are a bad choice of model for assigning semantics to terms; after all, it is the algebra that is restricting the behaviour of IA. As a consequence of having chosen IA as the semantic model, our algebra supports specifications of behaviours, which we claim are necessary for building scalable connectors. Furthermore, IA support a notion of

refinement which is a congruence with respect to a number of our composition operators. Accordingly, substitution (e.g., for connector reuse or evolution) can be performed compositionally. We have also defined a notion of equivalence over the terms of the algebra and, based on this, we established that the algebra is sound and complete. These properties advocate the adoption of IA as the semantic model for the algebra, based on how closely they align with the key dimensions of CONNECT that we specified in Section 1.

Our case study shows that the algebraic term representing a connector maps intuitively onto the models of the protocols to be mediated. The purpose of the algebra was to give system designers a high-level tool for specifying and reasoning about connector behaviours, which is why we favoured the utilisation of high-level primitives rather than developing yet another low-level process calculus. It seems that our high-level algebra allows a designer to easily and intuitively specify complex connectors, although further justification would be required for this claim based on further case studies.

We have not considered quantitative aspects of connectors as part of our algebra. As a minimum we would like to support time and probability. Although we have not introduced such aspects to our algebra yet, we have been looking at interactive Markov chains [21] and quantitative extensions of I/O automata [10,20]. Further work in this area involves looking at how these extensions may be carried across to IA. Besides having a quantitative model for expressing the semantics of our algebra, we would also need to consider how the syntax of our algebra would change. Clearly, primitives of the algebra will need to be annotated with quantitative values, but it will also be necessary to see whether it is meaningful to combine these values under the operators of the algebra.

In addition to IA, we are looking at modal specifications as a formalism for connectors [15]. These models are to some extent dissimilar to IA, yet they do share some common functionality [17]. We would like to compare and contrast these models with IA so that we can try to combine the best features of both for our algebra. Besides this future work, it is also our intention to determine whether the set of identified primitives is complete enough. Increasing the set of basic solutions should allow us to increase the types of behaviours that our algebra can capture. Discovery of such primitives is likely to come from analysis of further scenarios.

As broached at the end of Section 4, we need to take a closer look at automated connector synthesis. This is likely to be a definitive area on which our algebra is judged as to whether it has made a meaningful contribution to component-based design. We already have ideas relating to this in terms of rewriting systems [12], although the details require further fleshing out.

Thus, our preliminary algebra has raised a number of issues that we should take account of in formalising a comprehensive algebra to meet the demands imposed by CONNECT. Moving on from here, we intend to combine the positive features of our current algebra and refine its limitations in order to develop an algebra suitable for modelling connectors that reside in a truly heterogeneous world of ubiquitous devices.

References

1. CONNECT consortium. CONNECT Deliverable D2.1: Capturing functional and non-functional connector behaviours. CONNECT EU project no. 231167, http://connect-forever.eu/.
2. R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.
3. R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating Refinement Relations. In *CONCUR '98*, pages 163–178, London, UK, 1998. Springer-Verlag.
4. F. Arbab, T. Chothia, R. Mei, S. Meng, Y. Moon, and C. Verhoef. From coordination to stochastic models of QoS. In *COORDINATION '09*, pages 268–287, 2009.
5. F. Arbab, T. Chothia, S. Meng, and Y.-J. Moon. Component connectors with QoS guarantees. In *COORDINATION*, pages 286–304, 2007.
6. C. Baier. Probabilistic models for reo connector circuits. *Journal of Universal Computer Science*, 11(10):1718–1748, 2005.
7. C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *SCP*, 61(2):75–113, 2006.
8. P. Bhaduri and S. Ramesh. Interface synthesis and protocol conversion. *Form. Asp. Comput.*, 20(2):205–224, 2008.
9. S. Bliudze and J. Sifakis. The Algebra of Connectors - Structuring Interaction in BIP. *IEEE Transactions on Computers*, 57(10):1315–1330, 2008.
10. A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O Automata: A Complete Specification Theory for Real-time Systems. In *HSCC '10*, pages 91–100, 2010.
11. L. de Alfaro and T. A. Henzinger. Interface-based Design. In *In Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, 2005.
12. N. Dershowitz and J.-P. Jouannaud. *Rewrite Systems, Chapter 6 in Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics, pp. 243–320.* Elsevier and MIT Press, 1990.
13. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Prentice Hall, 2007.
14. P. Inverardi, V. Issarny, and R. Spalazzese. A Theory of Mediators for Eternal Connectors. In *In proceedings of ISOLA 2010*, 2010.
15. K. G. Larsen. Modal specifications. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, pages 232–246, London, UK, 1990. Springer-Verlag.
16. N. A. Lynch and M. R. Tuttle. An Introduction to Input/Output Automata. *CWI Quarterly*, 2:219–246, 1989.
17. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. Modal interfaces: unifying interface automata and modal specifications. In *EMSOFT '09*, pages 87–96, New York, NY, USA, 2009. ACM.
18. A. Schmitt and J.-B. Stefani. The kell calculus: A family of higher-order distributed process calculi. In *Global Computing*, pages 146–178, 2004.
19. R. Spalazzese and P. Inverardi. Mediating connector patterns for components interoperability. In *ECSA2010*, 2010, LNCS - To appear.
20. S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and Behaviors of Probabilistic I/O Automata. *Theor. Comput. Sci.*, 176(1-2):1–38, 1997.
21. D. N. Xu, G. Gössler, and A. Girault. Probabilistic Contracts for Component-based Design. In *ATVA '10 (to appear)*, 2010.