An Interface Theory for Service-Oriented Design

José Luiz Fiadeiro[1] and Antónia Lopes[2]

[1] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
jose@mcs.le.ac.uk

[2] Faculty of Sciences, University of Lisbon
Campo Grande, 1749–016 Lisboa, Portugal
mal@di.fc.ul.pt

Abstract. We revisit the notions of interface and component algebra proposed by de Alfaro and Henzinger in [7] for component-based design and put forward elements of a corresponding interface theory for service-oriented design. We view services as a layer that can be added over a component infrastructure and propose a notion of service interface for a component algebra that is an asynchronous version of relational nets adapted to SCA (the Service Component Architecture developed by the Open Service-Oriented Architecture collaboration).

1 Services vs. Components, Informally

In [7], de Alfaro and Henzinger put forward a number of important insights, backed up by mathematical models, that led to an abstract characterisation of essential aspects of component-based software design (CBD), namely in the distinction between the notions of component and interface, and the way they relate to each other. In this paper, we take stock on the work that we developed in the FET-GC2 integrated project SENSORIA [21] towards a language and mathematical model for service-oriented modelling [12], and investigate what abstractions can be put forward for service-oriented computing (SOC) that relate to the notions of interface and component algebra proposed in [7]. Our ultimate goal is similar to that of [7]: to characterise the fundamental structures that support SOC independently of the specific formalisms (Petri-nets, automata, process calculi, inter alia) that may be adopted to provide models for languages or tools.

A question that, in this context, cannot be avoided, concerns the difference between component-based and service-oriented design. The view that we adopt herein is that, on the one hand, services offer a layer of activity that can be superposed over a component infrastructure (what is sometimes referred to as a 'service overlay') and, on the other hand, the nature of the interactions between processes that is needed to support such a service overlay is intrinsically asynchronous and conversational, which requires a notion of component algebra that is different from the ones investigated in [7] for CBD.

The difference between components and services, as we see it, can be explained in terms of two different notions of 'composition', requiring two different notions of interface. In CBD, composition is integration-oriented — "the idea of component-based development is to industrialise the software development process by producing software applications by assembling prefabricated software components" [8]. In other words, CBD addresses what, in [10] we have called ‘physiological complexity’ — the ability to build a complex system by *integrating* a number of independently developed parts. Hence, interfaces for component-based design must describe the means through which software elements can be *plugged* together to build a product and the assumptions made by each element on the environment in which it will be deployed. Interfaces in the sense of [7] – such as assume/guarantee interfaces – fall into this category: they specify the combinations of input values that components implementing an interface must accept from their environment (*assumptions*) and the combinations of output values that the environment can expect from them (*guarantees*).

In contrast, services respond to the necessity for separating “need from the need-fulfilment mechanism” [8] and address what in [10] we have called ‘social complexity’: the ability of software elements to engage with other parties to pursue a given business goal. For example, we can design a seller application that may need to use an external supplier service if the local stock is low (*the need*); the discovery and selection of, and binding to, a specific supplier (*the need-fulfilment mechanism*) are not part of the design of the seller but performed, at run time, by the underlying middleware (service-oriented architecture) according to quality-of-service constraints. In this context, service interfaces must describe the properties that are provided (so that services can be discovered) as well as those that may be required from external services (so that the middleware can select a proper provider). The latter are not assumptions on the environment as in CBD — in a sense, a service *creates* the environment that it needs to deliver what it promises.

In the context of modelling and specifying services, one can find two different kinds of approaches — choreography and orchestration — which are also reflected in the languages and standards that have been proposed for Web services, namely WS-CDL for choreography and WS-BPEL for orchestration. In a nutshell, choreography is concerned with the specification and realizability of a ‘conversation’ among a (fixed) number of peers that communicate with each other to deliver a service, whereas orchestration is concerned with the definition of a (possibly distributed) business process (or workflow) that may use external services discovered and bound to the process at run time in order to deliver a service.

Whereas the majority of formal frameworks that have been developed for SOC address choreography (see [20] for an overview), the approach that we take in this paper is orchestration-oriented. More precisely, we propose to model the workflow through which a service is orchestrated as being executed by a network of processes that interact asynchronously and offer interaction-points to which clients and external services (executed by their own networks) can bind. Hence, the questions that we propose to answer are *What is a suitable notion of interface for such asynchronous networks of processes that deliver a service?*, and *What notion of interface composition is suitable for the loose coupling of the business processes that orchestrate the interfaces?*

The rest of this paper is technical and formal. In Section 2, we present a ‘component algebra’ that is a variation on relational nets [7] adapted to the Service Component Architecture [17]. This leads us to the characterisation of services as an ‘interface algebra’ (again in the sense of [7]), which we develop in Section 3. In Section 4, we compare our framework with formal models that have been proposed in the last few years for orchestration, namely [1,3,13].

2 A Service Component Algebra

As already mentioned, we adopt the view that services are delivered by systems of components as in SCA [17]:

> "SCA provides the means to compose assets, which have been implemented using a variety of technologies using SOA. The SCA composition becomes a service, which can be accessed and reused in a uniform manner. In addition, the composite service itself can be composed with other services [...] SCA service components can be built with a variety of technologies such as EJBs, Spring beans and CORBA components, and with programming languages including Java, PHP and C++ [...] SCA components can also be connected by a variety of bindings such as WSDL/SOAP web services, Java^{TM} Message Service (JMS) for message-oriented middleware systems and J2EE^{TM} Connector Architecture (JCA)".

In the terminology of [7], we can see components in the sense of SCA as implementing *processes* that are connected by *channels*. However, there is a major difference in the way processes are connected. In [7], and indeed many models used for service choreography and orchestration (e.g., [1,6,18]), communication is synchronous (based in I/O connections). In order to capture the forms of loose coupling that SOAs support, communication should be asynchronous: in most business scenarios, the traditional synchronous call-and-return style of interaction is simply not appropriate. This leads us to propose a model that is closer to communicating finite-state machines [4] (also adopted in [2]) than, say, I/O automata [15]. We call our (service) component algebra *asynchronous relational nets* (ARNs) to be consistent with [7].

In an asynchronous communication model, interactions are based on the exchange of messages that are transmitted through channels (wires in the terminology of SCA). For simplicity, we ignore the data that messages may carry. We organise messages in sets that we call *ports*. More specifically, every process consists of a (finite) collection of mutually disjoint ports, i.e. each message that a process can exchange belongs to exactly one of its ports. Ports are communication abstractions that are convenient for organising networks of processes as formalised below.

Every message belonging to a port has an associated *polarity*: $^-$ if it is an outgoing message (published at the port) and $^+$ if it is incoming (delivered at the port). This is the notation proposed in [4] and also adopted in [1].

Definition 1 (Ports and message polarity). *A port is a set of messages. Every port M has a partition $M^- \cup M^+$. The messages in M^- are said to have polarity $^-$, and those in M^+ have polarity $^+$.*

The actions of sending (publishing) or receiving (being delivered) a message m are denoted by $m!$ and m¡, respectively. In the literature, one typically finds $m?$ for the latter. In our model, we use $m?$ for the action of processing the message and m¿ for the action of discarding the message: processes should not refuse the delivery of messages but they should be able to discard them.

Definition 2 (Actions). *Let M be a port and $m \in M$.*

– *If* $m{\in}M^{-}$, *the set of actions associated with* m *is* $A_m = \{m!\}$ *and, if* $m{\in}M^{+}$, $A_m = \{m¡, m?, m¿\}$
– *The set of actions associated with* M *is* $A_M = \bigcup_{m \in M} A_m$.

In [7], predicates are used as a means of describing properties of input/output behaviour, i.e. establishing relations (or the lack thereof) between inputs and outputs of processes, leading to several classes of relational nets depending on when they are considered to be 'well-formed'. In the context of our asynchronous communication model, behaviour is observed in terms of the actions that are performed, for which the natural formalism to use is temporal logic (e.g., [14]). For simplicity, we use linear temporal logic, i.e. we observe traces of actions. In order not to constrain the environment in which processes execute and communicate, we take traces to be infinite and we allow several actions to occur 'simultaneously', i.e. the granularity of observations may not be so fine that we can always tell which of two actions occurred first.

Definition 3 (LTL). *Let* A *be a set (of actions). We use a classical linear temporal logic where every* $a{\in}A$ *is an atomic formula and formulas are interpreted over infinite traces* $\lambda{\in}(2^A)^\omega$. *For every collection* Φ *of formulas, we define:*

– $\Lambda_\Phi = \{\lambda{\in}(2^A)^\omega{:}\forall\phi{\in}\Phi(\lambda \models \phi)\}$
– $\Pi_\Phi = \{\pi{\in}(2^A)^*{:}\ \exists\lambda{\in}\Lambda_\Phi(\pi{\prec}\lambda)\}$ *where* $\pi{\prec}\lambda$ *means that* π *is a prefix of* λ. *Given* $\pi{\in}(2^A)^*$ *and* $B{\subseteq}A$, *we denote by* $(\pi{\cdot}B)$ *the trace obtained by extending* π *with* B.

We say that a collection Φ *of formulas entails* ϕ — $\Phi \models \phi$ — *iff* $\Lambda_\Phi{\subseteq}\Lambda_\phi$. *We say that a collection* Φ *of formulas is consistent iff* $\Lambda_\Phi \neq \emptyset$.

The fact that, at any given point i, it is possible that $\lambda(i)$ is empty means that we are using an open semantics, i.e. we are considering transitions during which the ARN is idle. This means that we can use the logic to reason about global properties of networks of processes, which is convenient for giving semantics to the composition of ARNs. Infinite traces are important because, even if the execution of individual processes in a service session is, in typical business applications, finite, the ARN may bind to other ARNs at run time as a result of the discovery of required services. Unbounded behaviour may indeed arise in SOC because of the intrinsic dynamics of the configurations that execute business applications, i.e. it is the configuration that is unbounded, not the behaviour of the processes and channels that execute in the configuration.

In this paper, we work with descriptions (sets of formulas) over different sets of actions, which requires that we are able to map between the corresponding languages:

Proposition and Definition 4 (Translation). *Let* $\sigma{:}A{\rightarrow}B$ *be a function. Given an LTL formula* ϕ *over* A, *we define its translation* $\sigma(\phi)$ *as the formula over* B *that is obtained by replacing every action* $a{\in}A$ *by* $\sigma(a)$. *The following properties hold:*

– *For every* $\lambda{\in}2^{B^\omega}$, $\lambda \models \sigma(\phi)$ *iff* $\sigma^{-1}(\lambda) \models \phi$ *where* $\sigma^{-1}(\lambda)(i)$ *is* $\sigma^{-1}(\lambda(i))$.
– *Any set* Φ *of LTL formulas over* A *is consistent if* $\sigma(\Phi)$ *is consistent.*
– *For every set* Φ *of LTL formulas over* A *and formula* ψ *also over* A, *if* $\Phi \models \psi$ *then* $\sigma(\Phi) \models \sigma(\psi)$.

Furthermore, if σ *is an injection, the implications above are equivalences.*

Proof. The first property is easily proved by structural induction, from which the other two follow. The properties of injections are proved in the same way on the direct image.

Notice that, in the case of injections, the translations induce conservative extensions, i.e. $\sigma(\Phi)$ is a conservative translation of Φ. We are particularly interested in translations that, given a set A and a symbol p, prefix the elements of A with '$p.$'. We denote these translations by $(p.\_)$. Note that prefixing defines a bijection between A and its image.

Definition 5 (Process). *A process consists of:*

- *A finite set γ of mutually disjoint ports.*
- *A consistent set Φ of LTL formulas over $\bigcup_{M\in\gamma} A_M$.*

Fig. 1 presents an example of a process $Seller$ with two ports. In the port depicted on the left, which we designate by L_{sl}, it receives the message buy and sends messages $price$ and $fwd\_details$. The other port, depicted on the right and called R_{sl}, has incoming message $details$ and outgoing message $product$. Among other properties, we can see that $Seller$ ensures to eventually sending the messages $product$ and $price$ in reaction to the delivery of buy. As explained below, the grouping of messages in ports implies that, whilst $price$ is sent over the channel that transmits buy, $product$ is sent over a different channel.

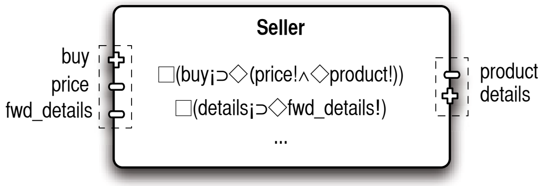

Fig. 1. Example of a process with two ports

Interactions in ARNs are established through channels. Channels transmit messages both ways, i.e. they are bidirectional, which is consistent with [4]. Notice that, in some formalisms (e.g., [2]), channels are unidirectional, which is not so convenient for capturing typical forms of conversation that, like in SCA, are two-way: a request sent by the sender through a wire has a reply sent by the receiver through the same wire (channel). This means that channels are agnostic in what concerns the polarity of messages: these are only meaningful within ports.

Definition 6 (Channel). *A channel consists of:*

- *A set M of messages.*
- *A consistent set Φ of LTL formulas over $A_M = \{m!, m¡ : m \in M\}$.*

Notice that in [2] as well as other asynchronous communication models adopted for choreography, when sent, messages are inserted in the queue of the consumer. In the context of loose coupling that is of interest for SOC, channels (wires) may have a behaviour of their own that one may wish to describe or, in the context of interfaces, specify. Therefore, for generality, we take channels as first-class entities that are responsible for delivering messages.

Channels connect processes through ports that assign opposite polarities to messages. Formally, the connections are established through what we call attachments:

Definition 7 (Connection). *Let M_1 and M_2 be ports and $\langle M, \Phi\rangle$ a channel. A connection between M_1 and M_2 via $\langle M, \Phi\rangle$ consists of a pair of bijections $\mu_i{:}M{\rightarrow}M_i$ such that $\mu_i^{-1}(M_i^+) = \mu_j^{-1}(M_j^-)$, $\{i,j\}{=}\{1,2\}$. Each bijection μ_i is called the* attachment *of $\langle M, \Phi\rangle$ to M_i. We denote the connection by $\langle M_1 \overset{\mu_1}{\leftarrow} M \overset{\mu_2}{\rightarrow} M_2, \Phi\rangle$.*

Proposition 8. *Every connection $\langle M_1 \overset{\mu_1}{\leftarrow} M \overset{\mu_2}{\rightarrow} M_2, \Phi\rangle$ defines an injection $\langle \mu_1, \mu_2\rangle$ from A_M to $A_{M_1}{\cup}A_{M_2}$ as follows: for every $m{\in}M$ and $\{i,j\}{=}\{1,2\}$, if $\mu_i(m){\in}M_i^-$ then $\langle \mu_1, \mu_2\rangle(m!) = \mu_i(m)!$ and $\langle \mu_1, \mu_2\rangle(m¡) = \mu_j(m)¡$.*

Definition 9 (Asynchronous relational net). *An asynchronous relational net (ARN) α consists of:*

- *A simple finite graph $\langle P, C\rangle$ where P is a set of nodes and C is a set of edges. Note that each edge is an unordered pair $\{p, q\}$ of nodes.*
- *A labelling function that assigns a process to every node and a connection to every edge such that:*
  - *If $p{:}\langle\gamma, \Phi\rangle$ and $q{:}\langle\gamma', \Phi'\rangle$ then $\{p, q\}$ is labelled with a connection of the form $\langle M_p \overset{\mu_p}{\leftarrow} M \overset{\mu_q}{\rightarrow} M_q, \Phi''\rangle$ where $M_p{\in}\gamma$ and $M_q{\in}\gamma'$.*
  - *For every $\{p, q\}{:}\langle M_p \overset{\mu_p}{\leftarrow} M \overset{\mu_q}{\rightarrow} M_q, \Phi\rangle$ and $\{p, q'\}{:}\langle M'_p \overset{\mu'_p}{\leftarrow} M' \overset{\mu'_{q'}}{\longrightarrow} M'_{q'}, \Phi'\rangle$, if $q \neq q'$ then $M_p \neq M'_p$.*

We also define the following sets:

- *$A_p = p.(\bigcup_{M\in\gamma_p} A_M)$ is the language associated with p,*
- *$A_\alpha = \bigcup_{p\in P} A_p$ is the language associated with α,*
- *$A_c{=}\langle p.\_{\circ}\mu_p, q.\_{\circ}\mu_q\rangle(A_M)$ is the language associated with $\gamma_c{:}\langle M_p \overset{\mu_p}{\leftarrow} M \overset{\mu_q}{\rightarrow} M_q\rangle$.*
- *Φ_α is the union of the following sets of formulas*
  - *For every $p{:}\langle\gamma, \Phi\rangle$, the prefix-translation Φ_p of Φ by $(p.\_)$.*
  - *For every $c{:}\langle M_p \overset{\mu_p}{\leftarrow} M \overset{\mu_q}{\rightarrow} M_q, \Phi\rangle$, the translation $\Phi_c{=}\langle p.\_ \circ \mu_p, q.\_ \circ \mu_q\rangle(\Phi)$*
- *$\Lambda_\alpha = \{\lambda{\in}{2^{A_\alpha}}^\omega{:}\ \forall p{\in}P(\lambda|_{A_p}{\in}\Lambda_{\Phi_p}) \wedge \forall c{\in}C(\lambda|_{A_c}{\in}\Lambda_{\Phi_c})\}$*
  The set of infinite traces that are projected to models of all processes and channels.
- *$\Pi_\alpha = \{\pi{\in}{2^{A_\alpha}}^*{:}\ \forall p{\in}P(\pi|_{A_p}{\in}\Pi_{\Phi_p}) \wedge \forall c{\in}C(\pi|_{A_c}{\in}\Pi_{\Phi_c})\}$*
  The set of finite traces that are projected to prefixes of models of all processes and channels.

We often refer to the ARN through the quadruple $\langle P, C, \gamma, \Phi\rangle$ where γ returns the set of ports of the processes that label the nodes and the pair of attachments of the connections that label the edges, and Φ returns the corresponding descriptions. The fact that the graph is simple — undirected, without self-loops or multiple edges — means that all interactions between two given processes are supported by a single channel and that no process can interact with itself. The graph is undirected because, as already mentioned, channels are bidirectional. Furthermore, different channels cannot share ports.

Notice that nodes and edges denote *instances* of processes and channels, respectively. Different nodes (resp. edges) can be labelled with the same process (resp. channel). Therefore, in order to reason about the properties of the ARN as a whole we need to translate the descriptions of the processes and channels involved to a language in which we can distinguish between the corresponding instances. The set Φ_α consists precisely of the translations of all the descriptions of the processes and channels using the nodes as prefixes for the actions that they execute. Notice that, by Prop. 4, these translations are conservative, i.e. neither processes nor channels gain additional properties because of the translations. However, by taking the union of all such descriptions, new properties may emerge, i.e. Φ_α is not necessarily a conservative extension of the individual descriptions.

A process in isolation, such as *Seller* (see Fig. 1), defines an ARN. Fig. 2 presents another ARN that also involves *Seller*. In this ARN, the port R_{sl} of *Seller* is connected with the port M_{sp} of process *Supplier*, which consists of the incoming message $request$ and the outgoing message $invoice$. The channel that connects R_{sl} and M_{sp} is described to be reliable with respect to $product$: it ensures to delivering $product$, which *Supplier* receives under the name $request$. Formally, this ARN consists of a graph with two nodes $sl{:}Seller$ and $sp{:}Supplier$ and one edge $\{sl, sp\}{:}w_{ss}$, where w_{ss} is the connection

$$\langle R_{sl} \xleftarrow{\mu_{sl}} \{m, n\} \xrightarrow{\mu_{sp}} M_{sp}, \{\Box(m! \supset \Diamond m¡)\}\rangle$$

with $\mu_{sl}=\{m \mapsto product, n \mapsto details\}$, $\mu_{sp}=\{m \mapsto request, n \mapsto invoice\}$.

The set $\Phi_{\text{SELLERWITHSUPPLIER}}$ consists of the translation of all properties of its processes and connections. Hence, it includes:

- $\Box(sl.buy¡ \supset \Diamond(sl.price! \wedge \Diamond sl.product!))$
- $\Box(sl.details¡ \supset \Diamond sl.fwd\_details!)$
- $\Box(sp.request¡ \supset \Diamond sp.invoice!)$
- $\Box(sl.product! \supset \Diamond sp.request¡)$

Notice that the last formula, which is the translation of the description of the channel, relates the languages of *Seller* and *Supplier*: the publication of $product$ by *Seller* leads to the delivery of $request$ to *Supplier*. In this context, $product$ and $request$ are just local names of the 'same' message as perceived by the two processes being connected. The ability to operate with local names is essential for SOC because, in the context of run-time discovery and binding, it is not possible to rely on a shared name space. This is why it is important that channels are first-class entities, i.e., that communication is established explicitly by correlating the actions that represent the local view that each party has of a message exchange.

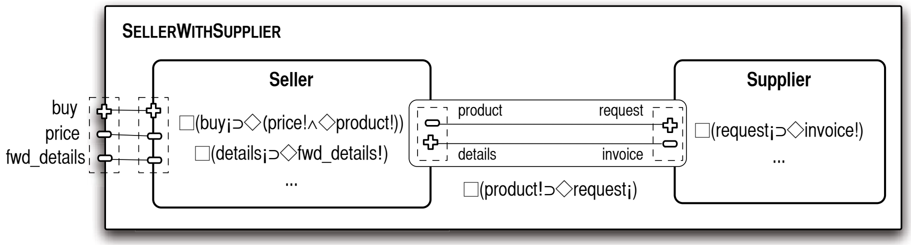

Fig. 2. An example of an ARN with two processes connected through a channel

In [7], joint consistency of the descriptions of the processes and the connections, i.e., of Φ_α, would be required for the ARN to be well defined. However, consistency does not ensure that the processes will always be able to make progress while interacting through the channels, which is why we prefer instead to use the following property as a criterion for well-formedness:

Definition 10 (Progress-enabled ARN). *We say that an ARN α is progress-enabled iff $\forall \pi \in \Pi_\alpha \exists A \subseteq A_\alpha (\pi \cdot A) \in \Pi_\alpha$.*

It is not difficult to see that any ARN α with a single process, such as $Seller$, is progress-enabled. This is because the process is isolated. In general, not every port of every process is necessarily connected to a port of another process. Such ports provide the points through which the ARN can interact with other ARNs. For example, SELLERWITHSUPPLIER has a single interaction point, which in Fig. 2 is represented by projecting the corresponding port to the external box.

Definition 11 (Interaction-point). *An interaction-point of an ARN $\alpha = \langle P, C, \gamma, \Phi \rangle$ is a pair $\langle p, M \rangle$ such that $p \in P$, $M \in \gamma_p$ and there is no edge $\{p, q\} \in C$ labelled with a connection that involves M. We denote by I_α the collection of interaction-points of α.*

Interaction-points are used in the notion of composition that we define for ARNs, which also subsumes the notion of interconnect of [7]:

Proposition and Definition 12 (Composition of ARNs). *Let $\alpha_1 = \langle P_1, C_1, \gamma_1, \Phi_1 \rangle$ and $\alpha_2 = \langle P_2, C_2, \gamma_2, \Phi_2 \rangle$ be ARNs such that P_1 and P_2 are disjoint, and a family $w^i = \langle M_1^i \overset{\mu_1^i}{\leftarrow} M \overset{\mu_2^i}{\rightarrow} M_2^i, \Psi^i \rangle$ $(i = 1 \ldots n)$ of connections for interaction-points $\langle p_1^i, M_1^i \rangle$ of α_1 and $\langle p_2^i, M_2^i \rangle$ of α_2 such that $p_1^i \neq p_1^j$ if $i \neq j$ and $p_2^i \neq p_2^j$ if $i \neq j$. The composition*

$$\alpha_1 \,\Big\|^{i=1\ldots n}_{\langle p_1^i, M_1^i \rangle, w^i, \langle p_2^i, M_2^i \rangle}\, \alpha_2$$

is the ARN defined as follows:

- *Its graph is $\langle P_1 \cup P_2, C_1 \cup C_2 \cup \bigcup_{i=1\ldots n} \{p_1^i, p_2^i\} \rangle$*
- *Its labelling function coincides with that of α_1 and α_2 on the corresponding subgraphs, and assigns to the new edges $\{p_1^i, p_2^i\}$ the label w^i.*

Proof. We need to prove that the composition does define an ARN. This is because we are adding to the sum of the graphs edges between interaction-points that do not share interaction-points, the resulting graph is simple. It is easy to check that the labels are well defined.

Fig. 2 can also be used to illustrate the composition of ARNs: SELLERWITHSUPPLIER is the composition of the two single-process ARNs defined by $Seller$ and $Supplier$ via the connection w_{ss}.

Given that we are interested in ARNs that are progress-enabled, it would be useful to have criteria for determining when a composition of progress-enabled ARNs is still progress-enabled. For this purpose, an important property of an ARN relative to its set of interaction-points is that it does not constrain the actions that do not 'belong' to the ARN. Naturally, this needs to be understood in terms of a computational and communication model in which it is clear what dependencies exist between the different parties. As already mentioned, we take it to be the responsibility of processes to publish and process messages, and of channels to deliver them. This requires that processes are able to buffer incoming messages, i.e., to be 'delivery-enabled', and that channels are able to buffer published messages, i.e., to be 'publication-enabled'.

Definition 13 (Delivery-enabled). *Let $\alpha=\langle P, C, \gamma, \Phi\rangle$ be an ARN, $\langle p, M\rangle \in I_\alpha$ one of its interaction-points, and $D_{\langle p,M\rangle}=\{p.m¡: m\in M^+\}$. We say that α is delivery-enabled in relation to $\langle p, M\rangle$ if, for every $(\pi\cdot A)\in \Pi_\alpha$ and $B\subseteq D_{\langle p,M\rangle}$, $(\pi\cdot B\cup(A\backslash D_{\langle p,M\rangle}))\in \Pi_\alpha$.*

The property requires that any prefix can be extended by any set of messages delivered at one of its interaction-points. Considering again the ARN defined by $Seller$, if its description is limited to the formulas shown in Fig. 1, then it is not difficult to conclude that the ARN is delivery-enabled for both its interaction-points — the constraints put on the delivery of *buy* and *details* are both satisfiable.

Definition 14 (Publication-enabled). *Let $h=\langle M,\Phi\rangle$ be a channel and $E_h = \{m! : m \in M\}$. We say that h is publication-enabled iff, for every $(\pi\cdot A)\in \Pi_\Phi$ and $B\subseteq E_h$, we have $\pi\cdot(B\cup(A\backslash E_h)) \in \Pi_\Phi$.*

The requirement here is that any prefix can be extended by the publication of a set of messages, i.e. the channel should not prevent processes from publishing messages. For example, the channel used in the connection w_{ss} is clearly publication-enabled: the extension of any prefix $(\pi\cdot A)$ in $\Pi_{\Phi_{w_{ss}}}$ with the publication of m (if it was not already in A) is still a prefix of an infinite trace that satisfies $\Box(m! \supset \Diamond m¡)$ by executing $m¡$ at a later stage.

Theorem 15. *Let $\alpha = (\alpha_1 \,\|^{i=1\ldots n}_{\langle p_1^i,M_1^i\rangle,w^i,\langle p_2^i,M_2^i\rangle}\, \alpha_2)$ be a composition of progress-enabled ARNs where, for each $i = 1\ldots n$, $w^i = \langle M_1^i \overset{\mu_1^i}{\leftarrow} M \overset{\mu_2^i}{\rightarrow} M_2^i, \Psi^i\rangle$. If, for each i=1...n, α_1 is delivery-enabled in relation to $\langle p_1^i,M_1^i\rangle$, α_2 is delivery-enabled in relation to $\langle p_2^i,M_2^i\rangle$ and $h^i=\langle M^i,\Phi^i\rangle$ is publication-enabled, then α is progress-enabled.*

We can use this theorem to prove that the ARN presented in Fig. 2 is progress-enabled. Because, as already argued, the channel used in this composition is publication-enabled and $Seller$ is delivery-enabled, it would remain to prove that so is $Supplier$, which is similar to the case of $Seller$.

Proposition 16. *Let $\alpha = (\alpha_1 \,\|^{i=1\ldots n}_{\langle p_1^i,M_1^i\rangle,w^i,\langle p_2^i,M_2^i\rangle}\, \alpha_2)$ be a composition.*

- *Let $\langle p_1', M_1'\rangle$ be an interaction-point of α_1 different from all $\langle p_1^i, M_1^i\rangle$. If α_1 is delivery-enabled in relation to $\langle p_1', M_1'\rangle$, so is α.*
- *Let $\langle p_2', M_2'\rangle$ be an interaction-point of α_2 different from all $\langle p_2^i, M_2^i\rangle$. If α_2 is delivery-enabled in relation to $\langle p_2', M_2'\rangle$, so is α.*

3 A Service Interface Algebra

In this section, we put forward a notion of interface for software components described in terms of ARNs and a notion of interface composition that is suitable for service-oriented design. As discussed in Section 1, this means that interfaces need to specify the services that customers can expect from ARNs as well as the dependencies that the ARNs may have on external services for providing the services that they offer.

In our model, a service interface identifies a number of ports through which services are provided and ports through which services are required (hence the importance of ports for correlating messages that belong together from a business point of view). Temporal formulae are used for specifying the properties offered or required.

Ports for required services include messages as sent or received by the external service. Therefore, to complete the interface we need to be able to express requirements on the channel through which communication with the external service will take place, if and when required. In order to express those properties, we need to have actions on both sides of the channel, for which we introduce the notion of dual port.

Definition 17 (Dual port). *Given a port M, we denote by M^{op} the port defined by $M^{op+} = M^-$ and $M^{op-} = M^+$.*

Notice that $(M^{op})^{op}$=M^{op}.

Definition 18 (Service interface). *A service interface i consists of:*

- *A set I (of interface-points) partitioned into two sets $I^{\rightarrow}$ and $I^{\leftarrow}$ the members of which are called the* provides- *and* requires-points*, respectively.*
- *For every interface-point r, a port M_r.*
- *For every point $r{\in}I^{\rightarrow}$, a consistent set of LTL formulas Φ_r over A_{M_r}.*
- *For every point $r{\in}I^{\leftarrow}$:*
  - *a consistent set of LTL formulas Φ_r over A_{M_r} making $\alpha_r = \langle\{r\}, \emptyset, M_r, \Phi_r\rangle$ delivery-enabled (see Def. 13),*
  - *a consistent set of LTL formulas Ψ_{M_r} over $\{m!, m¡\colon m{\in}M_r\}$ making $\langle M_r, \Psi_{M_r}\rangle$ a publication-enabled channel (see Def. 14).*

We identify an interface with the tuple $\langle I^{\rightarrow}, I^{\leftarrow}, M, \Phi, \Psi\rangle$ where $M_r{:}r{\in}I$, $\Phi_r{:}r{\in}I$, $\Psi_r{:}r{\in}I^{\leftarrow}$ are the indexed families that identify the ports and specifications of each point of the interface. Notice that different points may have the same port, i.e., ports are types.

The sets of formulas at each interface-point specify the protocols that services require from external services (in the case of requires-points) and offer to clients (in the case of provides-ports). For example, Fig. 3 presents a service interface with one provides and one requires-point (we use a graphical notation similar to that of SCA). On the left, we have an interaction point p through which the service is provided and, on the right side, the interaction point r through which an external service is required. According to what is specified, the service offers, in reaction to the delivery of the message buy, to reply by publishing the message $price$ followed eventually by $fwd\_details$. On the other hand, the required service is asked to react to the delivery of $product$ by publishing $details$.

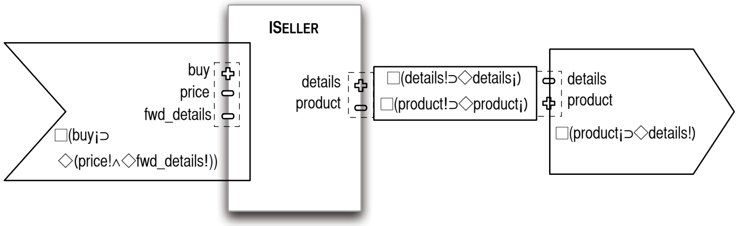

Fig. 3. An example of a service interface

The connection with the external service is required to ensure that the transmission of both messages is reliable.

Notice that the properties specified of the interface-points play a role that is different from the assumption/guarantee (A/G) specifications that have been proposed (since [16]) for networks of processes and also used in [19] for web services. The aim of A/G is to ensure compositionality of specifications of processes by making explicit assumptions about the way they interact with their environment. The purpose of the interfaces that we propose is, instead, to specify the protocols offered to clients of the service and the protocols that the external services that the service may need to discover and bind to are required to follow. This becomes clear in the definition of the notion of implementation of a service interface, which we call an orchestration. Compositionality is then proved (Theo. 22) under the assumptions made on the requires-points, namely delivery and publication enabledness, which concern precisely the way processes and channels interfere with their environments. That is, our interfaces do address the interference between service execution and their environment, but the formulas associated with requires-points are not assumptions and those of provides-ports are not guarantees in the traditional sense of A/G specifications.

Definition 19 (Orchestration). *An orchestration of a service interface $\langle I^{\rightarrow}, I^{\leftarrow}, M, \Phi, \Psi\rangle$ consists of:*

- *An ARN $\alpha = \langle P, C, \gamma, \Phi\rangle$ where P and I are disjoint, which is progress-enabled and delivery-enabled in relation to all its interaction-points.*
- *A one-to-one correspondence ρ between I and I_α; we will write $r \xrightarrow{\rho} p$ to indicate that $\rho(r) = \langle p, M_p\rangle$ for some port M_p.*
- *For every $r \in I^{\rightarrow}$, a polarity-preserving bijection $\rho_r : M_r \rightarrow M_p$ where $r \xrightarrow{\rho} p$.*
- *For every $r \in I^{\leftarrow}$, a polarity-preserving bijection $\rho_r : M_r^{op} \rightarrow M_p$ where $r \xrightarrow{\rho} p$.*

Let $\alpha^* = (\alpha \parallel_{\rho(r), w_r, \langle r, M_r\rangle} \alpha_r)_{r \in I^{\leftarrow}}$ with $w_r = \langle M_p \xleftarrow{\rho_r} M_r \xrightarrow{id} M_r, \Psi_r\rangle$, $r \xrightarrow{\rho} p$. We require that, for every $r \in I^{\rightarrow}$, $(p.\_ \circ \rho_r)^{-1}(\Lambda_{\alpha^*}) \subseteq \Lambda_{\Phi_r}$ — equivalently, $\Phi_{\alpha^*} \models p.(\rho_r(\Phi_r))$. A service interface that can be orchestrated is said to be consistent.

The condition requires that every model of the ARN composed with the requires-points and channels be also a model of the specifications of the provides-points. That is, no matter what the external services that bind to the requires-points do and how the channels transmit messages (as long as they satisfy the corresponding specifications), the

ARN will be able to operate and deliver the properties specified in the provides-points. Notice that, by Theo. 15 the composition is progress-enabled.

Also note that provides-points are mapped to interaction-points of the ARN preserving the polarity of the messages, but requires-points reverse the polarity. This is because every requires-point $r{\in}I^{\leftarrow}$ represents the external service that is required whereas $r\xrightarrow{\rho}p$ identifies the interaction-point through which that external service, once discovered, will bind to the orchestration. The ARNs α_r represent those external services.

Consider again the ARN defined by $Seller$ as in Fig. 1. It is not difficult to see that, together with the correspondences $p \mapsto \langle Seller, L_{sl}\rangle$ and $r \mapsto \langle Seller, R_{sl}\rangle$, $Seller$ defines an orchestration for the service interface ISELLER. Indeed, according to the definition above, $Seller^{*}$ is the composition

$$Seller \underset{\langle sl,R_{sl}\rangle,w_r,\langle r,R_{sl}^{op}\rangle}{||} \langle\{r\},\emptyset,R_{sl}^{op},\{\Box(product¡ \supset \Diamond details!)\}\rangle$$

Hence, the set $\Phi_{Seller^{*}}$ includes the following properties:

1. $\Box(sl.buy¡ \supset \Diamond(sl.price! \wedge \Diamond sl.product!))$ (from $Seller$)
2. $\Box(sl.details¡ \supset \Diamond sl.fwd\_details!)$ (from $Seller$)
3. $\Box(r.product¡ \supset \Diamond r.details!)$ (from the specification Φ_r of the requires-point)
4. $\Box(r.details! \supset \Diamond sl.details¡)$ (from the specification Ψ_r of the required channel)
5. $\Box(sl.product! \supset \Diamond r.product¡)$ (from Ψ_r)

It is not difficult to conclude that $\Box(p.buy¡ \supset \Diamond(sl.price! \wedge \Diamond sl.fwd\_details!)))$ is a logical consequence of $\Phi_{Seller^{*}}$. We just have to produce a chain of implications using (1), (5), (3), (4) and (2), in this order.

Definition 20 (Match). *A match between two interfaces* $i = \langle I^{\rightarrow}, I^{\leftarrow}, M^i, \Phi^i, \Psi^i\rangle$ *and* $j = \langle J^{\rightarrow}, J^{\leftarrow}, M^j, \Phi^j, \Psi^j\rangle$ *is a family of triples* $\langle r^m, s^m, \delta^m\rangle$, $m = 1\ldots n$, *where* $r^m{\in}I^{\leftarrow}$, $s^m{\in}J^{\rightarrow}$ *and* $\delta^m{:}M^i_{r^m}{\rightarrow}M^j_{s^m}$ *is a polarity-preserving bijection such that* $\Phi^j_{s^m} \models \delta^m(\Phi^i_{r^m})$. *Two interfaces are said to be compatible if their sets of interface-points are disjoint and admit a match.*

That is, a match maps certain requires-points of one of the interfaces to provides-points of the other in such a way that the required properties are entailed by the provided ones. Notice that, because the identity of the interface-points is immaterial, requiring that the sets of points of the interfaces be disjoint is not restrictive at all. We typically use $\delta^m{:}r^m{\rightarrow}s^m$ to refer to a match.

Definition 21 (Composition of interfaces). *Given a match* $\delta^m{:}r^m{\rightarrow}s^m$ *between compatible interfaces* $i = \langle I^{\rightarrow}, I^{\leftarrow}, M^i, \Phi^i, \Psi^i\rangle$ *and* $j = \langle J^{\rightarrow}, J^{\leftarrow}, M^j, \Phi^j, \Psi^j\rangle$, *their composition* $(i \;||_{\delta^m:r^m\rightarrow s^m}\; j) = \langle K^{\rightarrow}, K^{\leftarrow}, M, \Phi, \Psi\rangle$ *is defined as follows:*

- $K^{\rightarrow} = I^{\rightarrow} \cup (J^{\rightarrow} \setminus \{s^m : m = 1\ldots n\})$.
- $K^{\leftarrow} = J^{\leftarrow} \cup (I^{\leftarrow} \setminus \{r^m : m = 1\ldots n\})$.
- $\langle M, \Phi, \Psi\rangle$ *coincides with* $\langle M^i, \Phi^i, \Psi^i\rangle$ *and* $\langle M^j, \Phi^j, \Psi^j\rangle$ *on the corresponding points.*

Notice that the composition of interfaces is not commutative: one of the interfaces plays the role of client and the other of supplier of services.

We can now prove compositionality, i.e., that the composition of the orchestrations of compatible interfaces is an orchestration of the composition of the interfaces.

Theorem 22 (Composition of orchestrations). *Let* $i = \langle I^{\rightarrow}, I^{\leftarrow}, M^i, \Phi^i, \Psi^i \rangle$ *and* $j = \langle J^{\rightarrow}, J^{\leftarrow}, M^j, \Phi^j, \Psi^j \rangle$ *be compatible interfaces,* $\delta^m{:}r^m{\rightarrow}s^m$ *a match between them, and* $\langle \alpha, \rho \rangle$ *and* $\langle \beta, \sigma \rangle$ *orchestrations of* i *and* j*, respectively, with disjoint graphs.*

$$(\alpha \, \|^{m=1 \ldots n}_{\langle p^m, M_{p^m} \rangle, w^m, \langle q^m, M_{q^m} \rangle} \, \beta)$$

where $w^m = \langle M_{p^m} \xleftarrow{\rho_{r^m}} M_{r^m} \xrightarrow{\sigma_{s^m} \circ \delta^m} M_{q^m}, \Psi^i_{M_{r^m}} \rangle$, $\rho(r^m){=}\langle p^m, M_{p^m} \rangle$ *and* $\sigma(s^m) = \langle q^m, M_{q^m} \rangle$ *defines an orchestration of* $(i \, \|_{\delta^m : r^m \rightarrow s^m} \, j)$ *through the mapping* κ *that coincides with* ρ *on* I *and with* σ *on* J*.*

Compositionality is one of the key properties required in [7] for a suitable notion of interface. From the software engineering point of view, it means that there is indeed a separation between interfaces and their implementations in the sense that, at the design level, composition can be performed at the interface level independently of the way the interfaces will be implemented. In particular, one can guarantee that the composition of compatible interfaces can indeed be orchestrated, which is captured by the following corollary (the theorem provides a concrete way of deriving that orchestration from those of the component interfaces):

Corollary 23 (Preservation of consistency). *The composition of two compatible and consistent interfaces is consistent.*

4 Related Work and Concluding Remarks

In this paper, we took inspiration from the work reported in [7] on a theory of interfaces for component-based design to propose a formalisation of 'services' as interfaces for an algebra of asynchronous components. That is, we exposed and provided mathematical support for the view that services are, at a certain level of abstraction, a way of *using* software components — what is sometimes called a 'service-overlay' — and not so much a way of *constructing* software, which is consistent with the way services are being perceived in businesses [8] and supported by architectures such as SCA [17].

This view differs from the more traditional component-based approach in which components expose methods in their interfaces and bind tightly to each other (based on I/O-relations) to construct software applications. In our approach, components expose conversational, stateful interfaces through which they can discover and bind, on the fly, to external services or expose services that can be discovered by business applications. Having in mind that one of the essential features of SOC is loose-binding, we proposed a component algebra that is asynchronous — essentially, an asynchronous version of relational nets as defined in [7].

As mentioned in Section 1, most formal frameworks that have been proposed for SOC address *choreography*, i.e., the specification of a global conversation among a fixed number of peers and the way it can be realised in terms of the local behaviour generated by implementations of the peers. A summary of different choreography models that have been proposed in the literature can be found in [20]. Among those, we would like to distinguish the class of automata-based models proposed in [2,5,13], which are asynchronous. Such choreography models are inherently different from ours in the sense that they study different problems: the adoption of automata reflects the need to study the properties and realisability of conversation protocols captured as words of a language of message exchange. It would be tempting to draw a parallel between their notion of composite service — a network of machines — and our ARNs, but they are actually poles apart: our aim has not been to model the conversations that characterise the global behaviour of the peers that deliver a service, but to model the network of processes executed by an individual peer and how that network orchestrates a service interface for that peer — that is, our approach is *orchestration-based*. Therefore, we do not make direct usage of automata, although a reification of our processes could naturally be given in terms of automata. Our usage of temporal logic for describing ARNs, as a counterpart to the use of first-order logic in [7] for describing I/O communication, has the advantage of being more abstract than a specific choice of an automata-based model (or, for that matter, a Petri-net model [18]). This has also allowed us to adopt a more general model of asynchronous communication in which channels are first-class entities (reflecting the importance that they have in SOC). We are currently studying decidability and other structural properties of our model and the extent to which we can use model-checking or other techniques to support analysis.

Another notion of web service interface has been proposed in [3]. This work presents a specific language, not a general approach like we did in this paper, but there are some fundamental differences between them, for example in the fact that their underlying model of interaction is synchronous (method invocation), which is not suitable for loose coupling. The underlying approach is, like ours, orchestration-based but, once again, more specific than ours in that orchestrations are modelled through a specific class of automata supporting a restricted language of temporal logic specifications. Another fundamental difference is that, whereas in [3] the orchestration of a service is provided by an automaton, ours is provided by a network of processes (as in SCA), which provides a better model for capturing the dynamic aspects of SOC that arise from run-time discovery and binding: our notion of composition is not for integration (as in CBD) but for dynamic interconnection of processes. This is also reflected in the notion of interface: the interfaces used in [3] are meant for design-time composition, the client being statically bound to the invoked service (which is the same for all invocations); the interfaces that we proposed address a different form of composition in which the provider (the "need-fulfilment mechanism") is procured at run time and, therefore, can differ from one invocation to the next, as formalised in [11] in a more general algebraic setting.

Being based on a specific language, [3] explores a number of important issues related to compatibility and consistency that arise naturally in service design when one considers semantically-rich interactions, e.g., when messages carry data or are correlated according to given business protocols. A similar orchestration-based approach has been presented in [1], which is also synchronous and based on finite-state machines, and also addresses notions of compatibility and composition of conversation protocols (though, interestingly, based on branching time). We are studying an extension of our framework that can support such richer models of interaction (and the compatibility issues that they raise), for which we are using, as a starting point, the model that we adopted in the language SRML [12], which has the advantage of being asynchronous.

Although we consider that the main contribution of this paper is to put forward a notion of interface that can bring service-oriented design to the 'standards' of component-based design, there are still aspects of the theory of component interfaces developed in [7] that need to be transposed to services. For example, an important ingredient of that theory is a notion of compositional refinement that applies to interfaces (for top-down design) and a notion of compositional abstraction for implementations (orchestrations in the case of services), that can support bottom-up verification.

Other lines for further work concern extensions to deal with time, which is critical for service-level agreements, and to address the run-time discovery, selection and binding processes that are intrinsic to SOC. We plan to use, as a starting point, the algebraic semantics that we developed for SRML [11]. Important challenges that arise here relate to the unbounded nature of the configurations (ARNs) that execute business applications in a service-oriented setting, which is quite different from the complexity of the processes and communication channels that execute in those configurations.

Acknowledgments

We would like to thank Nir Piterman and Emilio Tuosto for many helpful comments and suggestions.

References

1. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing web service protocols. Data Knowl. Eng. 58(3), 327–357 (2006)
2. Betin-Can, A., Bultan, T., Fu, X.: Design for verification for asynchronously communicating web services. In: Ellis and Hagino [9], pp. 750–759
3. Beyer, D., Chakrabarti, A., Henzinger, T.A.: Web service interfaces. In: Ellis and Hagino [9], pp. 148–159
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
5. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: WWW, pp. 403–410 (2003)
6. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 2–17. Springer, Heidelberg (2007)
7. de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 148–165. Springer, Heidelberg (2001)
8. Elfatatry, A.: Dealing with change: components versus services. Commun. ACM 50(8), 35–39 (2007)
9. Ellis, A., Hagino, T. (eds.): Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14. ACM, New York (2005)

10. Fiadeiro, J.L.: Designing for software's social complexity. IEEE Computer 40(1), 34–39 (2007)
11. Fiadeiro, J.L., Lopes, A., Bocchi, L.: An abstract model of service discovery and binding. Formal Asp. Comput. (to appear)
12. Fiadeiro, J.L., Lopes, A., Bocchi, L., Abreu, J.: The Sensoria reference modelling language. In: Wirsing and Hoelzl
13. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. 328(1-2), 19–37 (2004)
14. Goldblatt, R.: Logics of time and computation. CSLI, Stanford (1987)
15. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
16. Misra, J., Chandy, K.M.: Proofs of networks of processes. IEEE Trans. Software Eng. 7(4), 417–426 (1981)
17. OSOA. Service component architecture: Building systems using a service oriented architecture (2005), White paper, `http://www.osoa.org`
18. Reisig, W.: Towards a theory of services. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 271–281. Springer, Heidelberg (2008)
19. Solanki, M., Cau, A., Zedan, H.: Introducing compositionality in web service descriptions. In: FTDCS, pp. 14–20. IEEE Computer Society, Los Alamitos (2004)
20. Su, J., Bultan, T., Fu, X., Zhao, X.: Towards a theory of web service choreographies. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 1–16. Springer, Heidelberg (2008)
21. Wirsing, M., Hoelzl, M. (eds.): Rigorous Software Engineering for Service-Oriented Systems. LNCS, vol. 6582. Springer, Heidelberg (2011)