Verifying A-Priori the Composition of Declarative Specified Services.

Federico Chesani, Paola Mello, Marco Montali, Paolo Torroni

***Abstract*—Service Oriented Architectures are knowing a wide success, thanks to the maturity of standards and implementations. Moreover, the possibility of composing complex systems starting from simpler services is becoming supported by industrial tools, although still immature at the standard level. However, the a-priori verification aspect, i.e. the capability of determining before executing the system if it exhibits some particular behaviour, is still matter of an intense research effort.**

In this paper we investigate the a-priori verification of bottom-up build systems from the behavioural viewpoint, where a choreography is not known at the beginning of the developing process, but rather it is verified only later. We focus on the problem of deciding if, given a set of services, there can be some fruitful interaction among them; if yes, we focus also on the problem of determining such interaction. Our approach is based on specifying the services by means of the ConDec declarative language, and by exploiting its translation to the SCIFF Framework to automatically perform the verification task.

***Index Terms*—ConDec Service Modeling, Declarative Languages, A-priori Verification, Logic Programming.**

I. Introduction

SERVICE ORIENTED COMPUTING emerged recently as an architectural paradigm for modeling and implementing business collaboration within and across organizational boundaries. The Web Service technology, currently the most advanced implementation of Service Oriented Architecture (SOA) principles, is almost established as the standard technology for current business implementations, thanks to the support it has received from the academics as well as from the industrial partners.

A key aspect in the success of Service Oriented Computing (SOC) is the possibility of composing different, heterogeneous services, yet achieving a complex system starting from more simple components. Interoperability at the level of data exchange, as well as at the level of service location and invocation, has been guaranteed by standards like WSDL [6].

Industrial tools are becoming available to support also the composition process, too. Although languages for defining composition rules and models have been proposed, but none of them has enjoyed the maturity level of the other standards. Initial proposals like BPMN [15], WS-CDL [7] and BPEL [3] have been criticized for their intrinsic procedural nature, while the need for open, declarative approaches has been recognized only later [4], [14].

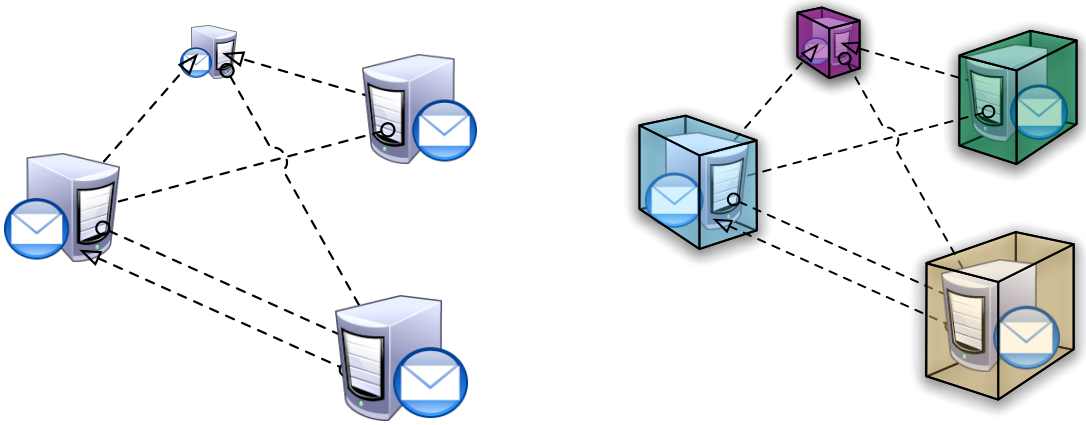
Automatic verification of properties regarding the behavioural aspects of the composed systems is deemed as a crucial step, but a comprehensive solution is lacking. Currently, the task of ensuring that two services can successfully cooperate is demanded to the software architect that is designing the system. Analogously, ensuring that the composed system will exhibit certain properties is a task that directly burden the developer. Although such guarantees can be verified by human users with small systems, there are serious doubts of achieving such results when the composed systems grow in dimension and interaction complexity. Hence, the task of automatically verifying a service composition a-priori (during the designing phase), is of the fundamental importance to foster the service composition and the "off-the-shelf" composition model .

Several approaches have been adopted to address the verification of service composition. A very common way consist of checking one service against a global description of a system, like in [1], [8]. In order to succeed, the following assumptions are usually made: 1) there is a description of the whole system, from a global point of view; 2) there is a description of the service under testing, such description not necessarily matching with the service internals; and 3) all services except the one under testing will behave as prescribed by the global specification (hence the global description can be used to reason upon the other services behaviour). Given this setting, the verification task determines if the behaviour of the service under testing is compatible with the global description (also named *choreography*). The choreography is intended as a sort of "legal, tight contract", and plays a double role: it specifies the "boundaries" for the service under testing, and provides the expected behaviour of the other (unknown) peers. The obvious advantage of such approach is that the verification task involves only a service description and a choreography: the component "certified" as compliant can be then adopted to play a certain role within the global system, independently of the other peers.

In this paper we investigate the verification of service composition from another viewpoint. We start from the assumption that, at least in the beginning steps, a global choreography is not defined (or is not yet available). Beside a "top-down" developing method, where the developer starts designing the choreography and proceeds to refine the components in several steps, there is also a common "bottom-up" projecting style, where the developer simply starts to build up her application by putting together the already available components. If this is the case, the models of all parties (called *local models* thereafter) are directly composed, in order to make them interact and mutually benefit from each other, as in Figure 1a.

The first problem we try to address is: *given a composition of services, does exist a successful interaction? If yes, how is made such interaction?* In case of a positive answer to the

DEIS - Department of Electronics, Informatics and Systems, University of Bologna. name.surname@unibo.it



(a) Bottom-Up

(b) Top-Down

Fig. 1. Developing complex systems using simpler services.

first question, we say that the models are *compatible*. Beside the yes/no answer, we deem as fundamental also knowing as much as possible about such interactions. E.g., knowing in advance the supported interaction traces would help in the analysis of the global system features and properties. Of course, "compatible" does not mean that a successful interaction will be effectively achieved at run-time.

A second problem we discuss in this work is about choreographies, intended no more as a tight contract to be respected, but rather as a set of constraints representing the desired properties of the system. In this view, choreographies are not intended as the set of requirements each service should fulfill to interact, but rather a set of desired features that the global system will exhibit. Roughly, this problem can be formulated as follow: *given a compatible set of services, does exists some successful interaction that honors the choreography constraints? If yes, how is made such interaction?* Also in this case we aim to know not only a yes/no answer, but also some sort of fully/partially specified interaction trace.

In our approach, services and choreographies are represented by means of a declarative language, and in particular using the ConDec language [13]. We agree with the critics moved to procedural approaches in [4], [14]. In particular, choreographies (intended as a set of properties or constraints of the final resulting system) are more naturally represented in terms of declarative rules/constraints, rather than by sentences of a procedural language. In this work, we extend the original ConDec model to the concepts of roles, and provide definition of service composition compatibility on the resulting ConDec models.

The ConDec semantics originally proposed by the authors is given as Linear Temporal Logic (LTL) formulas. Recently, a further semantics in terms of the SCIFF Framework [2] has been provided to ConDec [11]. Another contribution of this work is the definition of a method for automatically perform the verification tasks, by exploiting the SCIFF-based semantics, and its proof procedure.

In Section II we briefly introduce the ConDec language, aimed to declarative describe open processes/services. Then in Section III and in Section IV we try to better capture the notion of compatible services (compatible models of services) and of compliance to a choreography. In Section V we show how such properties are automatically verified exploiting the ConDec Language and its translation into the SCIFF Framework [2]. Finally we conclude and discuss future works.

II. Modeling a Service by Means of the ConDec Language

ConDec is a declarative, graphical language proposed by van der Aalst and Pesic [13] within the research field of Business Process Management (BPM). It aims to support specification, enactment and monitoring of Business Process, by means of constraints among activity executions. Constraints are declaratively expressed, as the authors claim that the adoption of a declarative approach fits better with complex, unpredictable processes, where a good balance between support and flexibility must be found. Although it has been originally proposed in the BPM context, it has been applied also in the far broader field of SOA. Former applications of ConDec regarded choreographies specification; in this paper, we adopt it to represent also service local models.

A. The ConDec Language

A ConDec model mainly consists of two parts: a set of activities, representing atomic units of work, and a set of relationships which constrain the way activities can be executed, and are therefore referred to as *constraints*. Constraints can be interpreted as policies/business rules, and may reflect different kinds of knowledge: external regulations and norms, internal policies and best practices, service/choreography goals, and so on.

Differently from procedural specifications, in which activities can be inter-connected only by means of control-flow relationships (sequence patterns, mixed with constructs supporting the splitting/merging of control flows), the ConDec language provides a number of control-independent abstractions to constrain activities, alongside the more traditional ones. In ConDec it is possible to insert past-oriented constraints, as

well as constraints that do not impose any ordering among activities.

Furthermore, while procedural specifications are closed, i.e., all what is not explicitly modeled is forbidden, ConDec models are *open*: activities can be freely executed, unless they are subject to constraints. This choice has two implications. First, a ConDec model accommodates many different possible executions, improving flexibility. Second, the language provides abstractions to explicitly capture not only what is mandatory, but also what is forbidden. In this way, the set of possible executions does not need to be expressed extensionally and models remain compact.

ConDec models do not impose a rigid scheduling of activities; instead, they leave the services free to execute activities in a flexible way, but respecting at the same time the imposed constraints. An execution trace, i.e. the set of the executed activities,we say that it is *supported* by a ConDec model if and only if it complies with all its constraints. Finally, it is important to note that well-defined ConDec models support only *finite* execution traces, because it must always be guaranteed that a BP will eventually terminate.

ConDec has been mapped to two different underlying logic frameworks, providing two different semantics for the ConDec constraints. Beside the originally proposed LTL mapping, in [11] a mapping to the SCIFF Framework has been proposed. SCIFF allows to define constraints in terms of the *happening* of events and *expectations* about the happening or the non-happening of other events. Events can be partially specified, by means of unbound variables. Possibly, such variables can be further constrained, a la CLP. E.g., it possible to say that if a certain event happens at time T, then another event is expected to happen at a time T' with the CLP constraint $T' > T$.

A simple ConDec model where activities a and b can be executed many times, but the execution of one automatically exclude the execution of the other, can be expressed in SCIFF by means of two rules (Integrity Constraints, using the SCIFF terminology): $\mathbf{H}(a, T) \Rightarrow \mathbf{EN}(b, T')$ and $\mathbf{H}(b, T) \Rightarrow \mathbf{EN}(a, T')$[1]. Note that such a simple model, if expressed using some procedural flow language such as for example Petri Nets, would lead to additional assumptions and choice points, thus making the final model pointlessly complicated.

Formally, a ConDec model $\mathcal{CM}$ is composed by a set of activities, which represent *atomic* units of work (i.e., units of work associated to single time points, and relations among activities, used to specify constraints on their execution. Optional constraints are also supported, to express preferable ways to interact, but allowing the possibility to violate them.

Definition 1 (ConDec model $\mathcal{CM}$)**.** A ConDec model is a triple $\langle \mathcal{A}, \mathcal{C}_m, \mathcal{C}_o \rangle$, where:

- $\mathcal{A}$ is a set of *activities*, represented as boxes containing their name;
- $\mathcal{C}_m$ is a set of *mandatory* constraints;
- $\mathcal{C}_o$ is a set of *optional* constraints.

Given a ConDec model $\mathcal{CM}$, notations $\mathcal{A}^{\mathcal{CM}}$, $\mathcal{C}_m^{\mathcal{CM}}$ and $\mathcal{C}_o^{\mathcal{CM}}$ respectively denote the set of activities, mandatory and optional constraints of $\mathcal{CM}$.□

[1]The two rules state that if a **H**appens, then b is **E**xpected **N**ot to happen, and viceversa.

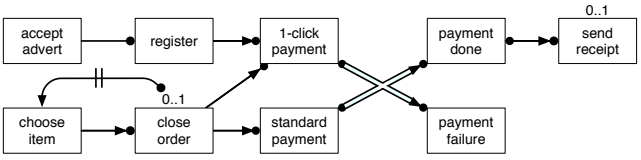


Fig. 2. A ConDec model.

B. A ConDec Example

Figure 2 shows the ConDec specification of a payment services. Boxes represent instances of activities. Numbers (e.g., 0; N..M) above the boxes are cardinality constraints that tell how many instances of the activity have to be done (e.g., never; between N and M). Edges and arrows represent relations(constraints) between activities. Double line arrows indicate alternate execution (after A, B must be done before A can be done again), while barred arrows and lines indicate negative relations (doing A disallows doing B). Finally, a solid circle on one end of an edge indicates which activity activates the relation associated with the edge. For instance, the execution of accept advert in Figure 2 does not activate any relation, because there is no circle on its end (a valid model could contain an instance of accept advert and nothing else); activity register instead activates a relation with accept advert (a model is not valid if it contains only register). If there is more than one circle, the relation is activated by each one of the activities that have a circle. Arrows with multiple sources and/or destinations indicate temporal relations activated/satisfied by either of the source/destination activities. The parties involved—a merchant, a customer, and a banking service to handle the payment—are left implicit.

In our example, the six left-most boxes are customer actions, payment done/ payment failure model a banking service notification about the termination status of the payment action, and send receipt is a merchant action. If register is done (once or more than once), then also accept advert must be done (before or after register) at least once. No temporal ordering is implied by such a relation. Conversely, the arrow from choose item to close order indicates that, if close order is done, choose item must be done at least once before close order. However, due to the barred arrow, close order cannot be followed by (any instance of) choose item. The 0..1 cardinality constraints say that close order and send receipt can be done at most once. 1-click payment must be preceded by register and by close order, whereas standard payment needs to be preceded only by close order (registration is not required). After 1-click or standard payment, either payment done or payment failure must follow, and no other payment can be done, before either of payment done/failure is done. After payment done there must be at most one instance of send receipt and before send receipt there must be at least a payment done. Sample valid models are: the empty model (no activity executed), a model containing one instance of accept advert and nothing else, and a model containing 5 instances of choose item followed

by a close order. A model containing only one instance of 1-click payment instead is not valid.

III. Compatibility and Legal Composition

In this section we address the problem of establishing if some local models are *compatible* [12], i.e. if there exists an interaction trace allowed by the composed system. We first try to establish if a single model does indeed support at least one interaction trace (i.e., it is *conflict-free*, and then we extend the notion to the composed system. Note that all the following definitions are based on the idea of execution traces, i.e. on a set of events (ground facts), happened at certain time point.

First of all, we introduce the notion of $\exists$-entailment: the aim is to define somehow when a model guarantees a property. To do so, we look at the traces allowed by the local ConDec model, and verify such property directly on the allowed traces. Once we move to verify a property on a trace, the semantic of the entailment symbol could be referred to the SCIFF semantic (as we do), as well as to the LTL semantics.

Definition 2 ($\exists$-entailment)**.** A property Ψ is $\exists$-entailed by a ConDec model $\mathcal{CM}$ ($\mathcal{CM} \models_\exists \Psi$) if at least one execution trace supported by $\mathcal{CM}$ entails the property as well. If that is the case, then one of the supported execution traces can be interpreted as an *example* which proves the entailment.

Definition 3 (Conflict-free model [12])**.** A ConDec model $\mathcal{CM}$ is *conflict-free* iff it supports at least one possible execution trace, i.e., iff

$$\mathcal{CM} \models_\exists true$$

A conflicting model is an over-constrained model: it is impossible to satisfy all its mandatory constraints at the same time.

Then we generalize the idea of conflict-freedom to the composed model:

Definition 4 (Composite model [12])**.** Given n ConDec models $\mathcal{CM}^i = \langle \mathcal{A}^i, \mathcal{C}^i_m, \mathcal{C}^i_o \rangle$, the *composite model* obtained by combining $\mathcal{CM}^1, \ldots, \mathcal{CM}^n$ is defined as :

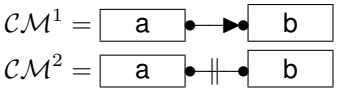
$$\text{COMP}\left(\mathcal{CM}^1, \ldots, \mathcal{CM}^n\right) \triangleq \langle \bigcup_{i=1}^{n} \mathcal{A}^i, \bigcup_{i=1}^{n} \mathcal{C}^i_m, \bigcup_{i=1}^{n} \mathcal{C}^i_o \rangle$$

Definition 5 (Compatibility)**.** Two ConDec models $\mathcal{CM}^1$ and $\mathcal{CM}^2$ are *compatible* if their composition is conflict-free, i.e., iff:

$$\text{COMP}\left(\mathcal{CM}^1, \mathcal{CM}^2\right) \models_\exists true$$

Obviously, the notion of compatibility can be generalized to the case of n local models. The detection of incompatibility means that a sub-set of the n local models leads to a conflict. note that checking compatibility could not be enough, as pointed out by the following example:

Example 1 (Trivial compatibility)**.** Two local models



have been composed. The two models are compatible, because they both support the empty execution trace; therefore, by carrying out solely a compatibility check would seem that a composition can be actually built. However, as soon as an activity is executed, $\mathcal{CM}^1$ and $\mathcal{CM}^2$ are contradictory: both activity a and activity b can not be executed in the composite model. In the general case, if none of the local models contains constraints which impose the execution of a certain activity (i.e., `existenceN`, `exactlyN` and `choice` constraints), compatibility always returns a positive answer, because the empty execution trace is supported. □

A. From Openness to Semi-Openness

Since a ConDec model is open, it implicitly allows the execution of activities not explicitly contained in the model itself. The following example clarifies the point. This characteristic could cause undesired compositions to be evaluated as correct, as in the following example.

Example 2 (Composition and openness issues)**.** A customer wants to find a seller to interact with. The customer comes with a ConDec model representing its own desired constraints and requirements. In particular, they express that:

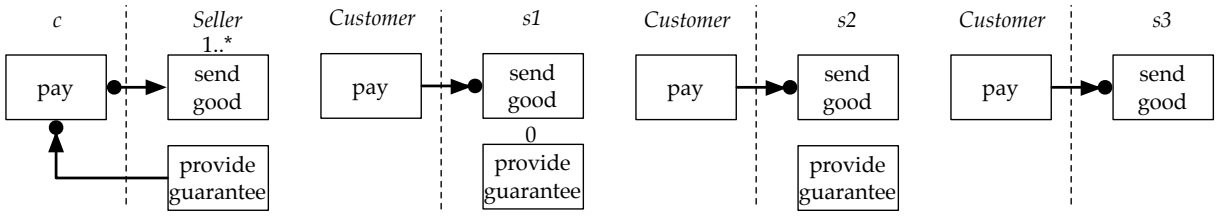
- the customer wants to receive a good from a seller;
- if the customer pays for a good, then she expects that the seller will deliver it;
- before paying, the customer wants the seller to provide a guarantee that the payment method is secure.

Figure 3 shows the ConDec graphical models ($\mathcal{CM}_C$) of the customer and of three candidate sellers. The three sellers differ for what concerns the possibility of emitting a guarantee upon request:

1) the seller depicted in Figure 3(b) ($\mathcal{CM}^1_S$) explicitly states that it does not provide any guarantee upon request;
2) the seller depicted in Figure 3(c) ($\mathcal{CM}^2_S$) explicitly supports the possibility of providing a guarantee;
3) the seller depicted in Figure 3(d) ($\mathcal{CM}^3_S$) does not mention provide guarantee among its activities.

Following Definition 5 checking compatibility between $\mathcal{CM}_C$ and the three candidate sellers would state that $\mathcal{CM}_C$ is not compatible with $\mathcal{CM}^1_S$, but it is compatible with $\mathcal{CM}^2_S$ and $\mathcal{CM}^3_S$. In particular, the two compositions $\mathcal{CM}_C \cup \mathcal{CM}^2_S$ and $\mathcal{CM}_C \cup \mathcal{CM}^3_S$ produce exactly the same global model. However, while the answer given for the first two compositions is in accordance with the intuitive notion of compatibility, the third one is not. In fact, when $\mathcal{CM}_C$ is composed with $\mathcal{CM}^2_S$, the behaviour of the seller is modified in that also the constraints of the customer must be respected. Contrariwise, when the composition between $\mathcal{CM}_C$ and $\mathcal{CM}^3_S$ is established, the local model of the customer has the effect of changing the local model of the seller, augmenting it with a new provide guarantee activity. During the execution, the customer would expect to receive a guarantee before paying, but this capability has not been mentioned by the seller in its local model, and therefore there could be the case that it is not supported. □

The example clearly shows that the openness assumption must be properly revised when dealing with the composition



(a) Constraints of the customer.

(b) Constraints of a seller which does not provide guarantees.

(c) Constraints of a seller which is able to provide a guarantee.

(d) Constraints of a seller which does not explicitly deal with the emission of a guarantee.

Fig. 3. Local models of a customer and of three candidate sellers.

problem. To ensure that a composition can be established, the obtained global model must obey to the following *semi-openness* requirement: for each involved party, *the activities under the responsibility of that party must also explicitly appear in its local model*.

B. Augmenting ConDec Models with Roles and Participants

In order to ensure the *semi-openness* assumption, each activity must be associated to its corresponding originator or role. The following definition extends the basic definition of a ConDec model with such a relationship.

Definition 6 (Augmented ConDec model)**.** An augmented ConDec model is a 4-tuple $\langle \mathcal{AO}, \mathcal{AR}, \mathcal{C}_m, \mathcal{C}_o \rangle$, where:

- $\mathcal{AO}$ is a set of (A, O) pairs where A is an activity and O is its originator;
- $\mathcal{AR}$ is a set of (A, R) pairs where A is an activity and R represents the role of its originator;
- $\mathcal{C}_m$ is a set of mandatory constraints over $\mathcal{AO}$ and $\mathcal{AR}$;
- $\mathcal{C}_o$ is a set of optional constraints over $\mathcal{AO}$ and $\mathcal{AR}$.

If $\mathcal{AR} = \emptyset$, the model is *completely grounded*. Contrariwise, if $\mathcal{AO} = \emptyset$ the model is *abstract*.

In this respect, a ConDec local model is defined as an augmented model containing also an indication about the identifier of the local model, and where an activity is associated either to such an identifier, or to an abstract role.

Definition 7 (Local augmented model)**.** A ConDec local augmented model is a 5-tuple $\langle ID, \mathcal{AO}, \mathcal{AR}, \mathcal{C}_m, \mathcal{C}_o \rangle$, where:

- ID is the identifier of the participant executing the local model;
- the other elements retain the meaning of Definition 6;
- $\mathcal{AO}$ is a set containing only elements of the type (A, ID).

A role identifies a *class* of originators; in the composition process, abstract roles employed in each local model are mutually grounded to concrete local models which participate to the composition.

Definition 8 (Grounding of a model)**.** Given an augmented model $\mathcal{CM}aug = \langle \mathcal{AO}, \mathcal{AR}, \mathcal{C}_m, \mathcal{C}_o \rangle$ and a function $\texttt{plays}$ mapping roles to concrete identifiers (i.e., stating that a certain identifier "plays" a given role), the grounding of $\mathcal{CM}aug$ on $\texttt{plays}$ is obtained by substituting each role R_i with the corresponding concrete participant identifier $\texttt{plays}(R_i)$:

- $\mathcal{AO} \downarrow_{\texttt{plays}} \triangleq \mathcal{AO} \cup \{(A, \texttt{plays}(R_i)) \mid R_i \in \texttt{dom}(\texttt{plays}) \wedge (A, R_i) \in \mathcal{AR}\}$;
- $\mathcal{AR} \downarrow_{\texttt{plays}} \triangleq \mathcal{AR} / \{(A, R_i) \mid R_i \in \texttt{dom}(\texttt{plays})\}$;
- $\mathcal{C}_m \downarrow_{\texttt{plays}}$ and $\mathcal{C}_o \downarrow_{\texttt{plays}}$ are updated accordingly.

If $\mathcal{AR} \downarrow_{\texttt{plays}} = \emptyset$, each role has been substituted by a concrete identifier and the model becomes ground. A legal composite ConDec can be now characterized as an augmented model obtained by composing a set of local models, each one grounded by taking into account the other ones, s.t. the composition is ground.

Definition 9 (Augmented composite model)**.** Given a set of augmented local models $\mathcal{L}^i = \langle ID_i, \mathcal{AO}^i, \mathcal{AR}^i, \mathcal{C}_m^i, \mathcal{C}_o^i \rangle$ $(i = 1, \ldots, n)$ and a function $\texttt{plays}$ mapping roles to identifiers, the composition of the local models w.r.t. $\texttt{plays}$ is defined as

$$\text{COMP}(\mathcal{L}^1, \ldots, \mathcal{L}^n)_{\texttt{plays}} = \bigcup_{i=1}^{n} \mathcal{L}^i \downarrow_{\texttt{plays}}$$

where the union of two augmented models is a shortcut representing the union of each corresponding element . A composition is *legal* iff $\text{COMP}(\mathcal{L}^1, \ldots, \mathcal{L}^n)_{\texttt{plays}}$ is ground (see Definition 6).

It is now possible to revise the notion of compatibility reflecting also the semi-openness assumption.

Definition 10 (Strong compatibility)**.** n local models $\mathcal{L}^i = \langle ID_i, \mathcal{AO}^i, \mathcal{AR}^i, \mathcal{C}_m^i, \mathcal{C}_o^i \rangle$ $(i = 1, \ldots, n)$ are *strong compatible* under $\texttt{plays}$ iff their augmented composition $\text{COMP}(\mathcal{L}^1, \ldots, \mathcal{L}^n)_{\texttt{plays}} = \langle \mathcal{AO}^{\cup}, \mathcal{AR}^{\cup}, \mathcal{C}_m^{\cup}, \mathcal{C}_o^{\cup} \rangle$ satisfies the following properties:

- $\text{COMP}(\mathcal{L}^1, \ldots, \mathcal{L}^n)_{\texttt{plays}}$ is legal;
- $\text{COMP}(\mathcal{L}^1, \ldots, \mathcal{L}^n)_{\texttt{plays}}$ is conflict-free;
- for each (a, ID_i) which belongs to $\mathcal{AO}^{\cup}$ but does not belong to $\mathcal{AO}^i$, it must hold that:

$$\text{COMP}(\mathcal{L}^1, \ldots, \mathcal{L}^n)_{\texttt{plays}} \models_{\forall} \texttt{absence}((a, ID_i))$$

The third point states that if a certain activity a has been associated to a participant ID_i, but ID_i has not explicitly mentioned a in its specification, then the composition must always ensure that a cannot be executed.

Example 3. Let us re-examine the compatibility between the local models of the customer and the second seller shown in Figure 3, supposing that their identifiers are respectively $alice$

and $hutter$, and $customer$ and $seller$ represent their roles. In the composition, $alice$ plays the role of $customer$, and $hutter$ plays the role of $seller$. Hence, $\texttt{plays}(alice) = customer$ and $\texttt{plays}(hutter) = seller$.

By adopting the definition of augmented models, the ConDec diagram of $alice$ is:

$$\begin{aligned}\mathcal{L}_{alice} = \langle &\{(pay, alice)\},\\ &\{(send\ good, seller), (provide\ guarantee, seller)\},\\ &\{\texttt{existenceN}(1, (send\ good, seller)), \ldots\},\\ &\emptyset\rangle\end{aligned}$$

The grounding of $alice$ w.r.t. the $\texttt{plays}$ function is $\mathcal{L}_{alice}\downarrow_{\texttt{plays}}=$

$$\begin{aligned}&\langle\{(pay, alice), (send\ good, hutter), (provide\ guarantee, hutter)\},\\ &\emptyset,\\ &\{\texttt{existenceN}(1, (send\ good, hutter)), \ldots\},\\ &\emptyset\rangle\end{aligned}$$

The grounding of $hutter$ is obtained in a similar way.

When the two local models are composed, the grounding of $alice$ causes $(provide\ guarantee, hutter)$ to belong to the set $\mathcal{AO}^{\cup}$ of the composition. Since the execution trace provide guarantee $\rightarrow$ pay $\rightarrow$ send good is compliant with the composition but $(provide\ guarantee, hutter) \notin \mathcal{AO}^{hutter}$, the two local models are not strong compatible. □

IV. Conformance to a choreography

Once a global model has been obtained through the composition step, and after a strong compatibility property has been verified, the application developer can move to further analise the resulting system, trying to understand if it entails some desired properties.

We represent a choreography as an augmented abstract ConDec model (i.e., an augmented model associating all the activities to roles and not to concrete participants – see Definition 6). When realizing a choreography with a set of concrete local models, different possible errors may arise:

- *Conflicting composition*: independently from the choreography, the chosen local models are not compatible.
- *Local non-conformance*: a concrete local model is not able to correctly play, within the choreography, the role it has been assigned to.
- *Global non-conformance*: even if each single local model is able to correctly play the role it has been assigned to, it could be the case that the global obtained model does not conforms the choreography anymore.

It could be also the case that a participant would not be able to play the role it has been assigned to, but it would anyway be able to take part to a conforming composition. Such a situation may arise because when the constraints of each local model are joint with the ones of the others, the constraints of the participant could be correctly "completed".

As a consequence it is necessary to first check that the composition is conflict-free, and then verify the whole composition against the choreography. To verify that a composition conforms to a desired choreography, two approaches can be followed. The *weak* approach states that the composition must be consistent with the choreography constraints in at least one supported execution, while the *strong* approach requires to guarantee that any execution supported by the composition respects the choreography.

Definition 11 (Weak conformance)**.** A composition of local models $\text{COMP}(\mathcal{L}_1, \ldots, \mathcal{L}_n)_{\texttt{plays}}$ is *weak conformant* with a choreography $\mathcal{C}hor$ iff:

- $\mathcal{L}_1, \ldots, \mathcal{L}_n$ are strong compatible w.r.t. $\texttt{plays}$ (see Definition 10);
- $\text{COMP}(\mathcal{L}_1, \ldots, \mathcal{L}_n)_{\texttt{plays}} \models_{\exists} \mathcal{C}hor \downarrow_{\texttt{plays}}$.

Definition 12 (Strong conformance)**.** A composition of local models $\text{COMP}(\mathcal{L}_1, \ldots, \mathcal{L}_n)_{\texttt{plays}}$ is *strong conformant* with a choreography $\mathcal{C}hor$ iff:

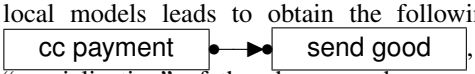
- $\mathcal{L}_1, \ldots, \mathcal{L}_n$ are strong compatible w.r.t. $\texttt{plays}$ (see Definition 10);
- $\text{COMP}(\mathcal{L}_1, \ldots, \mathcal{L}_n)_{\texttt{plays}} \models_{\forall} \mathcal{C}hor \downarrow_{\texttt{plays}}$.

Example 4 (Weak and strong conformance)**.** Let us consider a simple (fragment of a) choreography involving two roles – a customer and a seller. The choreography states that:

1) two possible payment methods are available to the customer (payment by credit card and payment by cash);
2) the customer can pay only after having closed the order;
3) if the customer pays, then the seller is entitled to send the ordered good and, conversely, a good is sent to the customer only if a payment has been previously done.

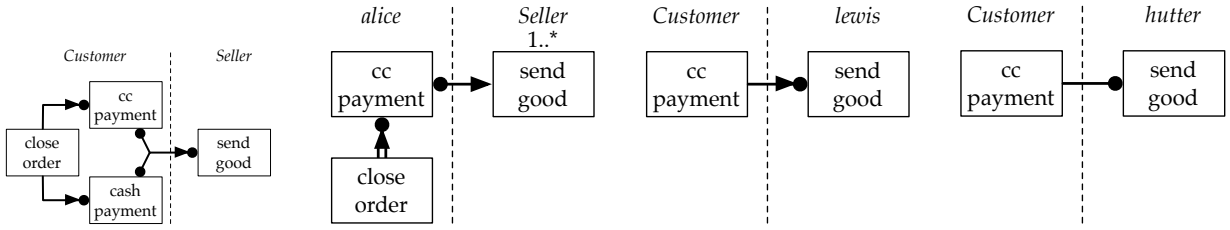
Figure 4 shows the ConDec model of the resulting choreography, and three possible local models which can be composed to realize such a choreography. In particular, $alice$ can play the role of $Customer$, while $hutter$ and $lewis$ can play the role of $Seller$.

Let us first consider the composition obtained by combining the model of $alice$ with the one of $lewis$. The composition is strong conformant with the choreography:

- The choreography allows an open choice on the payment modality, and both local models only deal with payment by credit card.
- The combination of the constraints which relate the payment with the delivery of the good in the two local models leads to obtain the following constraint 

, which is a "specialization" of the choreography one (no choice is present).
- $alice$ states that before paying, she wants to close the order, and that between two payments at least one close order must be executed; such a constraint is a specialization of the simple $\texttt{precedence}$ constraint contained in the choreography.

The composition obtained by combining the model of $alice$ with the one of $hutter$ is instead not strong conformant. In fact, $hutter$ does not impose any temporal ordering between the payment and the delivery of the good. Therefore, it could be possible that the good is sent twice: one time before the payment of alice, and another time afterwards. I.e., the following execution trace is supported by the composition: close order $\rightarrow$ send good $\rightarrow$ cc payment $\rightarrow$ send good. The first execution of the send good activity is not preceded by a payment, thus violating a prescription of the choreography.



(a) A simple payment choreography (b) Local model of $alice$. (c) Local model of $lewis$. (d) Local model of $hutter$.

Fig. 4. A simple choreography and three candidate local models (one customer and two sellers).

However, the composition is weak conformant, because it supports different possible executions which comply with the choreography. □

V. Verification through SCIFF

The SCIFF framework has been originally developed for the declarative specification and run-time verification of interaction protocols in the context of open Multi-Agent Systems [2]. It features:

- A rule-based language for modeling all the constraints that must be respected by the events characterizing the executions of the system under study. These rules relate the concepts of event occurrence with the one of expected/forbidden event, to model the (un)desired courses of interaction when a given situation is reached during the interaction. Events are modeled as logic programming terms (possibly containing variables), and are associated to an explicit execution/expected time; times and variables can be constrained by means of CLP constraints and Prolog predicates.
- A clear declarative semantics characterizing the execution traces compliant with the modeled rules.
- A corresponding proof procedure, sound and complete w.r.t. the declarative semantics, which is able to dynamically acquire the events occurring during a specific execution of the system, and check on-the-fly their compliance with the modeled rules.

In the last years, the framework has been applied in other contexts, such as clinical guidelines, business contracts and processes, service choreographies [9]. In particular, a complete mapping of all the ConDec constraints in terms of SCIFF rules has been provided, proving its soundness w.r.t. the original ConDec semantics (specified by means of LTL formulae) and discussing its impact on the verification techniques.

The SCIFF proof procedure has been then extended to deal also with the static verification of interaction protocols. g-SCIFF is the generative variant of SCIFF devoted to this task: instead of checking if a given (partial or complete) execution trace is compliant with the modeled rules, it is able to generate compliant execution traces or to return a negative answer if none exists [10]. In this way, g-SCIFF can be suitably exploited to effectively prove whether a ConDec model (translated to SCIFF) $\models_\exists$ or $\models_\forall$ a given property [9]. In particular, a ConDec model $\models_\exists$ a given property if and only if the g-SCIFF proof procedure is able to generate at least one execution trace compliant with the model and the property; such an execution trace can be considered as an example proving the existential entailment of the property. The case of $\models_\forall$ is reduced, similarly to model checking, to the $\models_\exists$ of the complemented property; the generation of an execution trace compliant with the model and the completed property can be considered as a counter-example showing that the original property is not entailed by the model in all its supported executions.

A. Compatibility Verification with g-SCIFF

Let us briefly describe how g-SCIFF carries out the compatibility verification between the customer's model shown in Figure 3(a) and each one of the three sellers modeled in Figures 3(b)-(d). As we have seen, the constraints of a composite model are obtained by joining all the constraints of the local models. Let us denote the composition of the customer's model with each seller's model with respectively $\mathcal{CM}_c^{s_1}$, $\mathcal{CM}_c^{s_2}$ and $\mathcal{CM}_c^{s_3}$. The first step, according to Definition 10, is to verify whether the composite model is legal; this is a syntactic test which can be trivially proven for all the three composite models: the role of *Customer* is grounded on c, and the role of *Seller* is respectively grounded on s_1, s_2 and s_3. The second and third step require instead the presence of a verifier able to prove conflict-freedom ($\models_\exists$) and to check if the composite model meet the semi-openness requirement ($\models_\forall$). When verifying the conflict-freedom of $\mathcal{CM}_c^{s_2}$ and $\mathcal{CM}_c^{s_3}$, g-SCIFF operates as follows:

- it starts from the $1..*$ constraint on send good, generating an occurrence of the event;
- this generated occurrence triggers the precedence ($\longrightarrow\!\!\bullet$) constraint involving send good and pay, leading to generate a previous payment;
- the generated payment, in turn, triggers the precedence ($\longrightarrow\!\!\bullet$) constraint involving pay and provide guarantee, leading to generate a previous emission of a guarantee.

At the end of the verification process, the following sample execution trace is therefore produced by g-SCIFF: provide guarantee → pay → send good [2]. When verifying the compatibility of $\mathcal{CM}_c^{s_1}$, instead, after the third step g-SCIFF realizes that the execution of provide guarantee clashes with

[2]The temporal relationships imposing the orderings between the three generated events are represented with CLP constraints

the 0 cardinality constraint (absence constraint) imposed by s_1, and returns a negative answer attesting the incompatibility of the local models.

The last requirement to be verified is that for each (a, ID_i) which belongs to $\mathcal{CM}_c^{s_2}$/$\mathcal{CM}_c^{s_3}$ but does not belong to the corresponding local model, it must hold that the absence of a is $\models_\forall$ by the composite model. In the case of $\mathcal{CM}_c^{s_2}$, no such activity actually exists, and therefore the requirement is directly met, attesting that the two local models are indeed compatible. Contrariwise, $\mathcal{CM}_c^{s_3}$ contains the activity (provide guarantee, s_3) which is however not contained in the local model of s_3. Therefore, g-SCIFF must prove whether all execution traces compliant with the composite model do not contain the execution of the provide guarantee activity. As already pointed out, $\models_\forall$ is reduced to $\models_\exists$ by complementing the property; in this specific case, the verification reduces to check whether at least one execution trace compliant with the composite model exists s.t. at least one execution of provide guarantee is contained in the trace. The execution trace provide guarantee $\rightarrow$ pay $\rightarrow$ send good, produced by g-SCIFF when checking the conflict-freedom of $\mathcal{CM}_c^{s_3}$, does satisfy the complemented property, and can be therefore considered as a counter-example showing that the semi-openness requirement is not met by the composite model, i.e., that c and s_3 are not compatible.

Finally, note that the conformance verification of a composite model with a choreography is carried out by g-SCIFF similarly to the case of compatibility. In the case of strong conformance, each constraint involved in the choreography is complemented and then separately checked w.r.t. $\models_\exists$. If at least one complemented property is $\models_\exists$, then the composition is not strong conforming with the choreography.

VI. Conclusions and Future Work

In this paper we have address some issues related to the process of composing complex system by using existing components, in the context of the Service Oriented Architectures. Our approach hypothesizes a bottom-up process in composing the global system, where the components are put together in order to achieve some interaction, and choreography constraints are taken into account only at a second stage. Peculiarities of our solution are 1) the use of a open, declarative, logic based language to represent models of the services and also the choreographies; 2) the exploit of the SCIFF Proof Procedure, and in particular of the g-SCIFF proof, to automatically perform all the verification tasks; and 3) beside a yes/no answer, our approach provides as output also some interaction trace that can be used to reason upon and analyse the global system.

This work is still in its preliminary stage, although some successful experimental results have been already obtained. Future works will be devoted to better assess the theory behind the solution, and to provide a better comparison with other approaches available in the literature, like [5], [8]. In particular, on the theme of the a-priori verification there is a huge research literature, as well as on the topic of reasoning on choreographies and roles. A further issue we intend to investigate is related to the possibility of introducing "soft" ConDec constraints: currently, constraints are considered as hard, and not respecting one constraint leads to an incompatibility response. We are hypothesizing situations where behavioural interface specifications can also comprehend compensations actions and explicit management of violations.

Acknowledgment

This work has been partially supported by the FIRB project TOCAI.it (RBNE05BFRK) and by the Italian MIUR PRIN 2007 project No. 20077WWCR8.

References

[1] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and M. Montali. An Abductive Framework for A-Priori Verification of Web Services. In A. Bossi and M. J. Maher, editors, *Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 39–50. ACM Press, 2006.

[2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable Agent Interaction in Abductive Logic Programming: the SCIFF framework. *ACM Transactions on Computational Logic*, 9(4), 2008.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.

[4] A. Barros, M. Dumas, and P. Oaks. A Critical Overview of the Web Services Choreography Description Language (WS-CDL). *BPTrends*, 2005.

[5] A. K. Chopra and C. P. Singh. Producing Compliant Interactions: Conformance, Coverage, and Interoperability. In *4th International Workshop on Declarative Agent Languages and Technologies IV (DALT 2006), Selected, Revised and Invited Papers*, volume 4327 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 2006.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.

[7] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Choreography Description Language Version 1.0, 2004. http://www.w3.org/TR/ws-cdl-10/.

[8] M. Baldoni and C. Baroglio and A. Martelli and V. Patti. A Priori Conformance Verification for Guaranteeing Interoperability in Open Environments. In A. Dan and W. Lamersdorf, editors, *Proocedings of the 4th International Conference on Service-Oriented Computing (ICSOC 2006)*, volume 4294 of *Lecture Notes in Computer Science*. Springer Verlag, 2006.

[9] M. Montali. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Framework*. PhD thesis, University of Bologna, 2009.

[10] M. Montali, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verification from Declarative Specifications Using Logic Programming. In M. G. D. L. Banda and E. Pontelli, editors, *ICLP*, number 5366 in LNCS, pages 440–454. Springer Verlag, 2008.

[11] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web - Accepted*, 2009.

[12] M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Controls to Users*. PhD thesis, Beta Research School for Operations Management and Logistics, Eindhoven, 2008.

[13] M. Pesic and W. M. P. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In *Proceedings of the BPM 2006 Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer Verlag, 2006.

[14] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed. Life After BPEL? In M. Bravetti, L. Kloul, and G. Zavattaro, editors, *Proceedings of the 2nd International Workshop on Web Services and Formal Methods (WS-FM 2005)*, volume 3670 of *Lecture Notes in Computer Science*, pages 35–50. Springer Verlag, 2005.

[15] S. A. White. Business Process Modeling Notation Specification 1.0. Technical report, OMG, 2006.