# Flexible Choreography-driven Service Selection

# Flexible Choreography-driven Service Selection

Matteo Baldoni, Cristina Baroglio, Elisa Marengo, Viviana Patti and Claudio Schifanella [*]
*Dipartimento di Informatica, Università degli Studi di Torino, c.so Svizzera 185, I-10149 Torino, Italy*
*E-mail: {baldoni,baroglio,emarengo,patti,schi}@di.unito.it*

**Abstract.** The greater and greater availability of services over the web motivates the growing interest in techniques that facilitate their re-use. A web service specification can be quite complex, including various operations and message exchange patterns. In this work, we propose a rule-based declarative representation of services, and in particular of WSDL operations, that enables the application of techniques for reasoning about actions and change, that are typical of agent systems. The representation allows reasoning on choreography roles and on possible role players, to the aim of selecting services which match in a flexible way with the specification. Flexible matches are an important tool that allows web service re-use but the proposals in the literature do not guarantee the preservation of those goals, that can be proved over the role specification. We show how to enrich various well-known matches so as to produce substitutions that preserve goals and that do not require service rollback. We also discuss the problem of the joint achievement of the individual goals of a group of choreography role players.

Keywords: Goal-driven reasoning, Semantic Matchmaking, Web Service Selection, Choreography

## 1. Introduction

Distributed applications over the World-Wide Web have obtained wide popularity. Uniform mechanisms have been developed for handling computing problems, which involve a large number of heterogeneous components, that are physically distributed and that interoperate. These developments coalesced around the web service paradigm [1] that, thanks to its platform-independent nature, allows enterprises to develop new business processes by combining existing services, of their own or retrieved over the web. For allowing the composition of a new business process, it is sufficient to have the public interfaces of services, without any need of disclosing internal implementations or the business logic. All these characteristics are particularly appealing in B2B scenarios, where it is desirable to have tools for creating cross-business applications, which require the least possible effort of harmonization of internal procedures.

Service selection plays an important role in the achievement of this result. In this respect, many advancements have been made. Semantic Web Services overcame the limits of the pure syntactical approaches, like UDDI, by adding to service descriptions a semantic layer, based on ontologies and rules. Further along this line, initiatives like OWL-S [32] and the Web Service Modeling Ontology (WSMO) [20] proposed richer annotations, aimed at supplying a semantic representation of the so called IOPEs (inputs, outputs, preconditions and effects of the service). These richer descriptions better support software re-use because they enable *flexible* forms of *matchmaking*, i.e., the retrieval of services whose descriptions do not exactly match with the corresponding queries— leading to the introduction of the concept of *degree of match* [33,29,20,9].

In many scenarios, however, the selection of single services, provided by the various partners, is not sufficient because services are to be coordinated, possibly avoiding to the parties the need to engage into expensive negotiations. Languages like WS-CDL [45] or WS-BPEL [30] fit this need, by allowing the defini-

[*]Corresponding author. E-mail: schi@di.unito.it.

tion of *patterns of interactions* in the form of choreographies (orchestrations, respectively), which represent the desired interaction from a global (or individual) perspective. In this context, a fundamental issue, that is typically left aside, is the definition of intelligent mechanisms for deciding whether to adopt a choreography. In our opinion, this choice should encompass two intertwined factors: (i) the possibility of achieving goals of interest when performing a role in a choreography, and (ii) the already mentioned flexible matchmatching capabilities. For instance, consider a medical centre which needs to decide whether becoming a check-up center in a consortium, asking its associates to act according to a given choreography. The decision would depend on whether the operations it provides fit the choreography specification, and also on whether the participation will still allow it to achieve some goal of its own interest, e.g. to ask patients to collect results at the office rather than on-line. By reasoning on the role specification the centre can check whether this goal is achievable.

What happens, however, when the specification is instantiated by using service operations which do not precisely match with it? Will the desired goals remain achievable? For instance, what if a specific service that plays the role of booking centre offers to the patients the additional option to receive results in their e-mail? What will the impact on the medical centres, which play the complementary role in the interaction, be? Current (flexible, semantic) matchmaking techniques work on a single operation at a time. Since each operation can influence the executability and the outcomes of the subsequent ones in the choreography, in general, a sequence of individually identified operations might not work [6,8].

This work addresses the above issues by adopting the *agent paradigm* [35,42,5], and by showing that choreographies provide a precise context for the selection of services, that should be taken into account during the matchmaking process. The agent paradigm offers proper abstractions to articulate a model of the interaction among services in two different levels: the choreography level, where shared patterns of interaction are described from a global and public perspective, and the interaction policy level, where the interactive behavior of the single service is represented from a local perspective. Agents show the ability of performing goal-driven forms of reasoning, and they also show autonomy and pro-activity, which are helpful characteristics when dealing with open environments [14,17,37,19].

*Original Contribution.* We define a *declarative framework* which: (i) provides reasoning techniques that support a goal-driven selection of choreography roles; (ii) enables the definition of an enhanced notion of flexible semantic matchmaking, in the context provided by a choreography. The representation is based on the rule-based language described in [4,3]. The description of roles and of service policies builds upon WSDL2 exchange patterns, represented as atomic actions with preconditions and effects.

This framework, allows for reasoning on operations, on goal achievement, and on choreographies, and it allows the design of services with a much higher degree of autonomy with respect to the existing ones and whose behavior resembles more closely the behavior of autonomous agents. Reasoning itself becomes a basis for fostering the re-use of services, by identifying also services which match only to some degree with the specifications [46,41]. Since services play roles for achieving goals, we introduce the notion of *conservative* substitution, to denote substitutions that preserve the goals of interest. We also characterize the class of flexible matches that are conservative.

Each party involved in a choreography may have its own goals, that it tries to pursue. Parties individually check for the possibility to achieve the goals of their own interest, identifying subsets of choreography enactments they will try to perform. One further question to answer, therefore, is: will such individually retrieved enactments be compatible? We discuss the issue and draft a simple technique that allows the joint achievement of the parties goals, without the need of disclosing them.

*Organization.* Section 2 sets the representation of services and of choreographies that we adopt, and explains how it is possible to reason on such a representation to allow each service to check the reachability of its goals locally, i.e. by using only the specification of the desired choreography role. Section 3 discusses various kinds of match, that can be applied for selecting the service operations that come in handy for implementing the specifications supplied by the choreography, and reports our proposal for verifying whether a match is *conservative*. Section 4 tackles the joint achievement of personal goals. A running example concerning a healthcare system is used to better explain the proposed notions and mechanisms. Conclusions and related works end the paper.

## 2. Reasoning about services

This section introduces the notation used to represent services and discusses the problem of verifying a global goal. The notation is based on a logical theory for reasoning about actions and change in a *modal logic programming setting* and on the language *DYnamics in LOGic*[1], described in details in [4]. This language is designed for specifying agents behaviour and for modeling dynamic systems. It is fully set inside the logic programming paradigm by defining programs by sets of Horn-like rules and giving a SLD-style[2] proof procedure. The capability of reasoning about interaction protocols, supported by the language, has already been exploited for customizing web service selection and composition w.r.t. to the user's constraints, based on a semantic description of the services [4]. The language is based on a modal theory of actions and mental attitudes where modalities are used for representing primitive and complex actions as well as the agent beliefs. Complex actions are defined by inclusion axioms [2] and by making use of action operators from dynamic logic, like sequence ";" and test "?".

In this framework, the problem of reasoning amounts either to build or to traverse a sequence of transitions between *states*. A state is a set of *fluents*, i.e., properties whose truth value can change over time, due to the application of actions. In general, we cannot assume that the value of each fluent in a state is known: we want to have both the possibility of representing unknown fluents and the ability of reasoning about the execution of actions on incomplete states. To explicitly represent unknown fluents, we use an epistemic operator $\mathbf{B}$, to represent the beliefs an entity has about the world: $\mathbf{B}f$ means that the fluent $f$ is known to be true, $\mathbf{B}\neg f$ means that the fluent $f$ is known to be false. A fluent $f$ is undefined when both $\neg\mathbf{B}f$ and $\neg\mathbf{B}\neg f$ hold ($\neg\mathbf{B}f \wedge \neg\mathbf{B}\neg f$): this is represented by the operator $\mathbf{U}$. Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*. Moreover, the epistemic operator $\mathbf{M}$ is defined as the dual of $\mathbf{B}$, i.e. $\mathbf{M}f$ is $\neg\mathbf{B}\neg f$. $\mathbf{M}f$ means that $f$ is considered to be possible. For the sake of readability, we will add as a superscript of $\mathbf{B}$ and $\mathbf{M}$ the name of the service that has the belief.

Services exhibit interfaces, called port-types, which make a set of operations available to possible clients. In our proposal, a *service description* is defined as a pair $\langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O}$ is a set of basic operations, and $\mathcal{P}$ (*policy*) is a description of the complex behavior of the service. Analogously to what happens for OWL-S composite processes, $\mathcal{P}$ is built upon basic operations and tests, that control the flow of execution.

### *2.1. Basic operations*

The main languages for representing web services identify different kinds of operations. According to WSDL and OWL-S, for instance, there are four main kinds of operations [1] (or atomic processes, in OWL-S terminology [32]). In this work we consider both standard operations and message exchange patterns as defined in WSDL 2.0 [3]), which include but are not limited to the four cases of OWL-S:

- *in-only (*or *one-way)* involves a single message exchange, a client invokes an operation by sending a message to the service;
- *out-only (*or *notify)* involves a single message exchange, the client receives a message from the service;
- *in-out (*or *request-response)* involves the exchange of two messages, initiated by the invoker of the operation, which sends a message to the service and then waits for a response;
- *out-in (*or *solicit-response)* involves the exchange of two messages, the order of the messages is inverted w.r.t. a request-response: first the invoker waits for a message from the service and then it sends an answer.
- *in-optional-out* involves the exchange of one or two messages. The operation is initiated by the invoker which sends a message to the service and then may receive an optional response;
- *out-optional-in* involves the exchange of one or two messages. First the invoker waits for a message from the supplier and then it can send an optional answer.

A basic operation is described in terms of its *executability preconditions* and *effects*. The former is a set of fluents (introduced by the keyword **possible if**) which must be contained in the service state in order for the operation to be applicable. The latter is a set of fluents (introduced by the keyword **causes**) which

[1]http://www.di.unito.it/∼alice/dynamicslogic/index.html
[2]Selective Linear Definite clause resolution.

[3]More details can be found in http://www.w3.org/TR/wsdl20-adjuncts/ and http://www.w3.org/TR/wsdl20-additional-meps/

will be added to the service state after the operation execution. Formally, the syntax is:

$$\text{operation}(content)\ \textbf{possible if}\ P_s \qquad (1)$$

$$\text{operation}(content)\ \textbf{causes}\ E_s \qquad (2)$$

$E_s$ and $P_s$ respectively denote the fluents which are expected as effect of the execution of the operation, and the precondition to its execution; *content* denotes possible additional data that is required by the operation. Notice that an operation can also be implemented as an invocation to some other service.

When executed, operations trigger a revision process on the actor's beliefs. Since we describe web services from a *subjective* point of view, we distinguish the case when the service is the initiator (the operation *invoker*) from the one in which it is the servant (the operation *supplier*) by further decorating the operation name with a notation inspired by [10]: $operation^{\gg}$ denotes the operation from the point of view of the invoker, while $operation^{\ll}$ denotes the operation from the point of view of the supplier. The two views are *complementary*, so if one view includes the act of sending a message, the other correspondingly includes the act of receiving the message.

Operations are defined in terms of their inputs, outputs, preconditions, and effects, as usual for semantic web services [32]. In the case of the *invoker*, preconditions $P_s$ in (1) and effects $E_s$ in (2) are respectively the conditions required by the operation for being invoked, and the expected effects of its execution. In the case of the *supplier*, we represent the conditions that enable the executability of the operation and the side effects of its execution. In order to distinguish them from the above, we use $R_s$ instead of $P_s$ in (1) to denote preconditions, calling them *requirements*. To denote *side effects* we use $S_s$ instead of $E_s$ in (2).

For example, a *buy* operation of a selling service has as a precondition the fact that the invoker has a valid credit card, as inputs the credit card number of the buyer and its expiration date, as output it generates a receipt, and as effect the credit card is charged. From the point of view of the supplier, the requirement to the execution is to have an active connection to the bank, and the side effect is that the store availability is decreased while the service bank account is increased of the perceived amount.

Let us now introduce the formal representation of basic operations and of message exchange patterns. For each case we report both views. Inputs and outputs are represented as single messages for simplicity

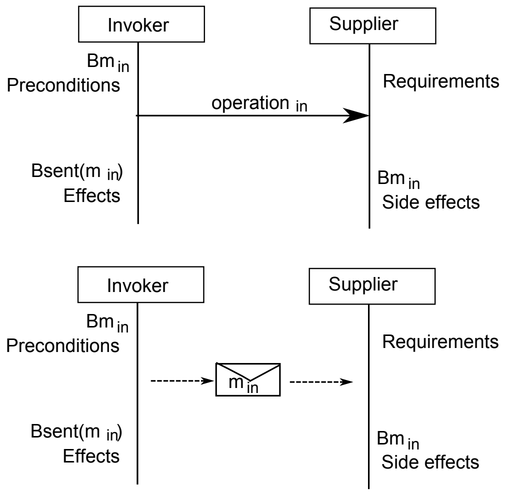

Fig. 1. The in-only basic operation: UML sequence diagram (top) representation and corresponding message exchange (bottom).

but the representation can easily be extended to sets of exchanged data, as in Example (2.1).

### 2.1.1. *In-only*

The *invoker* (see Figure 1) requests an execution which involves sending an information $m_{in}$ to the supplier; the invoker must know the information to send ($\mathbf{B}^{Invoker} m_{in}$) before the invocation, see (a) in Table 1. The invoker can execute the operation only if the preconditions to the operation ($P_s$) are satisfied in its current state (a). The execution of the invocation brings about the effects $E_s$ of the operation (c), and the invoker will know ($\mathbf{B}^{Invoker} sent(m_{in})$) that it has sent an information to the supplier (b). Using OWL-S terminology, $m_{in}$ is the *input* of the operation, while $P_s$ and $E_s$ are its preconditions and effects. One-way operations have no output.

On the other hand, the *supplier*, which exhibits the one-way operation as one of the services that it can execute, has the requirements $R_s$ (d). The execution of the operation causes the fact that the supplier will know ($\mathbf{B}^{Supplier} m_{in}$) the information sent by the invoker (e). We also allow the possibility of having some side effects ($S_s$) on the supplier's state (f). These are not to be confused with the operation effects described by IOPE, and were added for the sake of completeness.

### 2.1.2. *Out-only*

The *invoker* requests an execution which involves receiving an information $m_{out}$ from the supplier. The invoker can execute the operation only if the preconditions $P_s$ to the operations are satisfied in its current

| **Operation** | **Views** | **Representation** |
|---|---|---|
| *In-only* | Invoker | (a) operation$^{\gg}_{in}(m_{in})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$ |
| *(One-Way)* | | (b) operation$^{\gg}_{in}(m_{in})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$ |
| | | (c) operation$^{\gg}_{in}(m_{in})$ **causes** $E_s$ |
| | Supplier | (d) operation$^{\ll}_{in}(m_{in})$ **possible if** $R_s$ |
| | | (e) operation$^{\ll}_{in}(m_{in})$ **causes** $\mathbf{B}^{Supplier} m_{in}$ |
| | | (f) operation$^{\ll}_{in}(m_{in})$ **causes** $S_s$ |
| *Out-only* | Invoker | (a) operation$^{\gg}_{out}(m_{out})$ **possible if** $P_s$ |
| *(Notify)* | | (b) operation$^{\gg}_{out}(m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$ |
| | | (c) operation$^{\gg}_{out}(m_{out})$ **causes** $E_s$ |
| | Supplier | (d) operation$^{\ll}_{out}(m_{out})$ **possible if** $R_s$ |
| | | (e) operation$^{\ll}_{out}(m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{out}$ |
| | | (f) operation$^{\ll}_{out}(m_{out})$ **causes** $\mathbf{B}^{Supplier} sent(m_{out})$ |
| | | (g) operation$^{\ll}_{out}(m_{out})$ **causes** $S_s$ |
| *In-out* | Invoker | (a) operation$^{\gg}_{io}(m_{in}, m_{out})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$ |
| *(Request-* | | (b) operation$^{\gg}_{io}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$ |
| *response)* | | (c) operation$^{\gg}_{io}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$ |
| | | (d) operation$^{\gg}_{io}(m_{in}, m_{out})$ **causes** $E_s$ |
| | Supplier | (e) operation$^{\ll}_{io}(m_{in}, m_{out})$ **possible if** $R_s$ |
| | | (f) operation$^{\ll}_{io}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{in}$ |
| | | (g) operation$^{\ll}_{io}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{out}$ |
| | | (h) operation$^{\ll}_{io}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} sent(m_{out})$ |
| | | (i) operation$^{\ll}_{io}(m_{in}, m_{out})$ **causes** $S_s$ |
| *Out-in* | Invoker | (a) operation$^{\gg}_{oi}(m_{in}, m_{out})$ **possible if** $P_s$ |
| *(Solicit-* | | (b) operation$^{\gg}_{oi}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$ |
| *response)* | | (c) operation$^{\gg}_{oi}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{in}$ |
| | | (d) operation$^{\gg}_{oi}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$ |
| | | (e) operation$^{\gg}_{oi}(m_{in}, m_{out})$ **causes** $E_s$ |
| | Supplier | (f) operation$^{\ll}_{oi}(m_{in}, m_{out})$ **possible if** $R_s$ |
| | | (g) operation$^{\ll}_{oi}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{out}$ |
| | | (h) operation$^{\ll}_{oi}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} sent(m_{out})$ |
| | | (i) operation$^{\ll}_{oi}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{in}$ |
| | | (l) operation$^{\ll}_{oi}(m_{in}, m_{out})$ **causes** $S_s$ |

Table 1

Representation of the first four basic operations; for each of them we report both the invoker's and the supplier's view.

state (a). The execution brings about the effects $E_s$ of the operation (c), and the invoker will know the received information (b). $m_{out}$ is the *output* of the operation, while $P_s$ and $E_s$ are its preconditions and effects. Out-only operations have no input. The *supplier* must meet the requirements $R_s$ (d). The execution of the operation causes the supplier to know the message to send (e) and that it has sent some information to the invoker (f). The message $m_{out}$ the supplier sends to the invoker is built during the internal execution of the operation. Finally, (g) accounts for possible side effects on the supplier's state.

### 2.1.3. *In-out*

The *invoker* requests an execution which involves sending an information $m_{in}$ (the operation input) and then receiving an answer $m_{out}$ from the supplier (the operation output).

The invoker can execute the operation only if the precondition $P_s$ is satisfied in its current state and if it owns the information to send (a) (see Table 1). The invocation brings about the effects $E_s$ (d), and the fact that the invoker knows it has sent the input $m_{in}$ to the supplier (b). One further effect is that the invoker knows the answer returned by the operation (c). This representation abstracts away from the actual message

| **Operation** | **Views** | **Representation** |
|---|---|---|
| *In-optional-out* | Invoker | (a) operation$^{\gg}_{io?}(m_{in}, m_{out})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$ |
| | | (b) operation$^{\gg}_{io?}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$ |
| | | (c) operation$^{\gg}_{io?}(m_{in}, m_{out})$ **causes** $\mathbf{M}^{Invoker} m_{out}$ |
| | | (d) operation$^{\gg}_{io?}(m_{in}, m_{out})$ **causes** $E_s$ |
| | Supplier | (e) operation$^{\ll}_{io?}(m_{in}, m_{out})$ **possible if** $R_s$ |
| | | (f) operation$^{\ll}_{io?}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{in}$ |
| | | (g) operation$^{\ll}_{io?}(m_{in}, m_{out})$ **causes** $\mathbf{M}^{Supplier} m_{out}$ |
| | | (h) operation$^{\ll}_{io?}(m_{in}, m_{out})$ **causes** $\mathbf{M}^{Supplier} sent(m_{out})$ |
| | | (i) operation$^{\ll}_{io?}(m_{in}, m_{out})$ **causes** $S_s$ |
| *Out-optional-in* | Invoker | (a) operation$^{\gg}_{oi?}(m_{in}, m_{out})$ **possible if** $P_s$ |
| | | (b) operation$^{\gg}_{oi?}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$ |
| | | (c) operation$^{\gg}_{oi?}(m_{in}, m_{out})$ **causes** $\mathbf{M}^{Invoker} m_{in}$ |
| | | (d) operation$^{\gg}_{oi?}(m_{in}, m_{out})$ **causes** $\mathbf{M}^{Invoker} sent(m_{in})$ |
| | | (e) operation$^{\gg}_{oi?}(m_{in}, m_{out})$ **causes** $E_s$ |
| | Supplier | (f) operation$^{\ll}_{oi?}(m_{in}, m_{out})$ **possible if** $R_s$ |
| | | (g) operation$^{\ll}_{oi?}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} m_{out}$ |
| | | (h) operation$^{\ll}_{oi?}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Supplier} sent(m_{out})$ |
| | | (i) operation$^{\ll}_{oi?}(m_{in}, m_{out})$ **causes** $\mathbf{M}^{Supplier} m_{in}$ |
| | | (l) operation$^{\ll}_{oi?}(m_{in}, m_{out})$ **causes** $S_s$ |

Table 2
Representation of basic operations with optional messages.

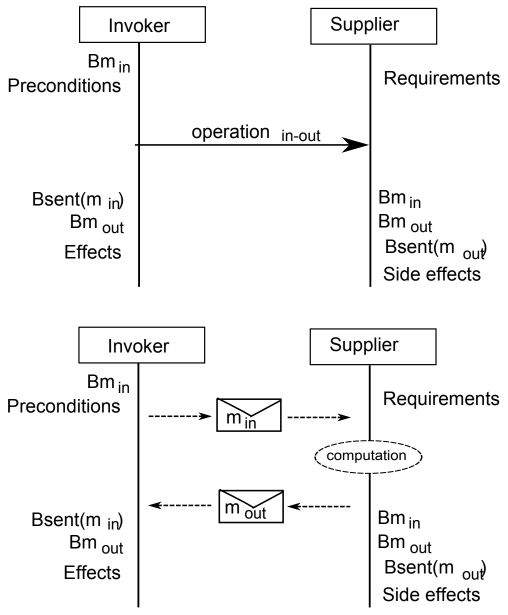

Fig. 2. The in-out basic operation: UML sequence diagram (top) representation and corresponding message exchange (bottom).

exchange mechanism, which is implemented. The aim is to reason on the effects of the execution on the mental state of the parties [3].

As for one-way operations, the *supplier* has the requirements $R_s$ to the operation execution (e). It receives an input $m_{in}$ from the invoker (f). The execution of the operation produces an answer $m_{out}$ (g), which is sent to the invoker (h). As usual, possible side effects on the supplier's state are accounted for(i). On the supplier's side, we can notice more evidently the abstraction of the representation from the actual execution process. In fact, we do not model how the answer is produced but only the fact that it is produced.

### 2.1.4. Out-in

The *invoker* requests an execution which involves receiving an information $m_{out}$ (the operation output) and then sending a message $m_{in}$ to the supplier (the input). The operation can be invoked only if the precondition $P_s$ is satisfied in its current state (a). The invocation brings about the effects $E_s$ (e). The invoker receives a message $m_{out}$ from the supplier (b) then, it produces the input information $m_{in}$ which is sent to the supplier, see (c) and (d). As for notify operations, the *supplier* must fulfill the requirements $R_s$ (f). The execution of the operation causes the supplier to know the information to send (g) and that it has sent such information to the invoker (h). Moreover, it produces also the knowledge on the information $m_{in}$ received from the invoker (i). As above, we allow the possibility of having some side effects on the supplier's state (l).

## *2.2. In-optional-out and Out-optional-in*

The *In-optional-out* and *Out-optional-in* operations share the same representation with the corresponding *In-out* and *Out-in* operations: the only difference is the use of the epistemic operator $\mathbf{M}$ to represent optional messages (see Table 2). We also slightly change the decorations to visually remark the difference with exchange patterns with no optionality. So we use $\mathsf{operation}^{\gg}_{io?}$ and $\mathsf{operation}^{\ll}_{io?}$ to respectively denote the invoker's and the supplier's views of in-optional-out operations, while we use $\mathsf{operation}^{\gg}_{oi?}$ and $\mathsf{operation}^{\ll}_{oi?}$ to denote the invoker's and the supplier's views of out-optional-in operations.

**Example 2.1 Healthcare reservation service running example (I part).** *We use a running example taken from the healthcare domain, nowadays one of the most interesting and growing application fields for service technology [11,43]. Medical centres more and more frequently get associated into consortia, offering to patients comprehensive sets of services. When this happens, patients no longer interact with the single structure but through a unified interface which, behind the scenes, relies on the services and on the capabilities provided by the various associates. The interesting point is that medical centers do not need to harmonize their procedures, as long as the interface of the services they provide fits the specifications supplied by the unified reservation system.*

*Let us define* searchExamination*, an operation of kind in-out, offered by a healthcare reservation system. This operation can be invoked by a* patient *to search for information about available time slots for booking a medical examination in registered medical centres. From the point of view of the patient, the operation is:*

*(a)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ $\mathbf{possible\ if}\ \ \mathbf{B}^{patient}examination \wedge \mathbf{B}^{patient}\neg listObtained$
*(b)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ $\mathbf{causes}\ \ \mathbf{B}^{patient}sent(examination)$
*(c)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ $\mathbf{causes}\ \ \mathbf{B}^{patient}exmList$
*(d)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ $\mathbf{causes}\ \ \mathbf{B}^{patient}listObtained$

*The input of the operation is $examination$, its output is $exmList$, i.e. the list of the available dates. The precondition $P_s$ is the belief* $\mathbf{B}^{patient}\neg$ listObtained *while the $E_s$ is the belief* $\mathbf{B}^{patient}$ listObtained.

*From the point of view of the healthcare system, instead, the operation is represented as:*

*(a)* $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$
$\quad$ $\mathbf{possible\ if}\ \ true$
*(b)* $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$
$\quad$ $\mathbf{causes}\ \ \mathbf{B}^{hrr}examination$
*(c)* $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$
$\quad$ $\mathbf{causes}\ \ \mathbf{B}^{hrr}exmList$
*(d)* $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$
$\quad$ $\mathbf{causes}\ \ \mathbf{B}^{hrr}sent(exmList)$

*In this case the sets $R_s$ and $S_s$ of requirements and side effects are empty. The operation expects as input the requested medical examination, and it produces an $exmList$, which is sent to the patient, so after the operation the belief $\mathbf{B}^{hrr}sent(exmList)$ will be in its belief state.*

## *2.3. Service policies and choreography roles*

The framework also accounts for *behaviors* that require the execution of many operations. A behavior is a collection of clauses of kind:

$$p_0\ \ \mathbf{is}\ \ p_1, \ldots, p_n \tag{3}$$

where $p_0$ is the name of the procedure and each $p_i$, $i = 1, \ldots, n$, is either an operation, a test action (denoted by the symbol ?), or a procedure call. Procedures can be recursive and are executed in a goal-directed way, similarly to standard logic programs. Their definitions can be non-deterministic as in Prolog. Behaviors also allow the representation of choreography roles.

We represent a *choreography* $\mathcal{C}$ as a tuple $(R_1, \ldots, R_n)$ of interacting and complementary *roles*, each role $R_i$ being a subjective view of the encoded interaction. A role $R$ is represented by a pair $\langle \mathcal{O}, \mathcal{P} \rangle$, similarly to services. The difference with services is that it composes *specifications* of operations and not implemented operations. The player of each role must supply appropriate implementations. We call such specifications *unbound operations* and formally represent them as basic operations, in terms of their preconditions and effects. This is one of the advantages of adopting a logic language: it allows handling implementations and specifications in the same way.

**Example 2.2 Healthcare reservation (II part).** *As an instance, let us consider the* bookExamination *pro-*

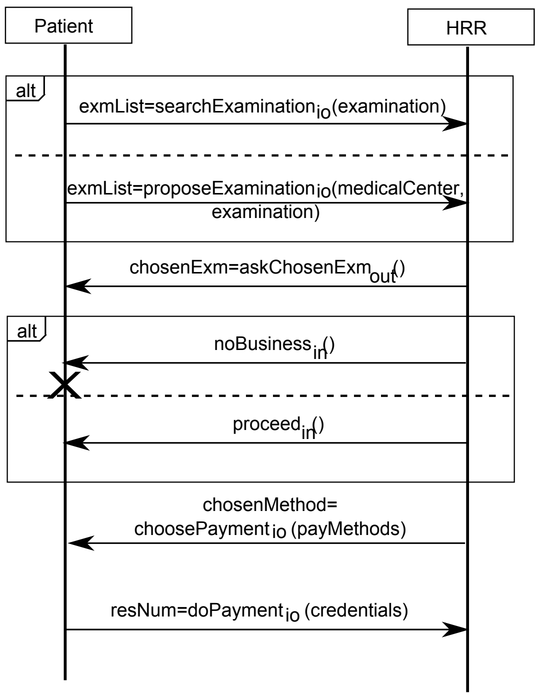
Patient
HRR
alt
exmList=searchExamination io(examination)
exmList=proposeExamination io(medicalCenter, examination)
chosenExm=askChosenExm out()
alt
noBusiness in()
proceed in()
chosenMethod= choosePayment io (payMethods)
resNum=doPayment io (credentials)

Fig. 3. Example of a simple interaction protocol, for booking a medical examination, expressed as a UML sequence diagram.

*cedure of the healthcare reservation system, as implemented by a possible patient service.*

*(a)* bookExamination **is**
  $?(\mathbf{U}medicalCenter \vee \mathbf{B}\neg medicalCenter)$;
  $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$;
  $\mathsf{askChosenExm}^{\ll}_{out}(chosenExm)$;
  evaluateAndReserve.
*(b)* bookExamination **is**
  $?(\neg\mathbf{U}medicalCenter)$;
  $\mathsf{proposeExamination}^{\gg}_{io}((medicalCenter,$
  $examination), exmList)$
  $\mathsf{askChosenExm}^{\ll}_{out}(chosenExm)$;
  evaluateAndReserve.
*(c)* evaluateAndReserve **is**
  $\mathsf{noBusiness}^{\ll}_{in}()$.
*(d)* evaluateAndReserve **is**
  $\mathsf{proceed}^{\ll}_{in}()$;
  $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$;
  $\mathsf{doPayment}^{\gg}_{io}((credential, payInfo), resNum)$.

*If the patient does not have a specific medical centre to look for ($?(\mathbf{U}medicalCenter \vee \mathbf{B}\neg medicalCenter)$), she invokes an operation for searching a list of available places and slots (*$\mathsf{searchExamination}^{\gg}_{io}$*). Instead, if she is interested in a specific medical center ($?(\neg\mathbf{U}medicalCenter)$), e.g. because it is covered by her medical insurance, she invokes a different operation provided by the healthcare reservation role (*$\mathsf{proposeExamination}^{\gg}_{io}$*). This returns only the slots available at the specified medical center. After that, the patient waits for being asked about her choice (*$\mathsf{askChosenExm}^{\ll}_{out}$*): if the patient returns to the booking service an empty selecttion, the interaction is interrupted (*$\mathsf{noBusiness}^{\ll}_{in}$*), otherwise it continues with the *$\mathsf{proceed}^{\ll}_{in}$* message followed by the selection of the payment method (*$\mathsf{choosePayment}^{\ll}_{io}$*). Finally, the patient performs the payment (*$\mathsf{doPayment}^{\gg}_{io}$*). In Appendix 6 the reader can find all the missing operations.*

## 2.4. Reasoning on goals

In the outlined framework, it is possible to reason about goals by means of queries of the form

$$Fs \;\; \mathbf{after} \;\; p \qquad (4)$$

where $Fs$ is the *goal* (represented as a conjunction of fluents), that should hold after the execution of the policy $p$. Checking if a formula of this kind holds amounts to answering the query: "Is it possible to execute $p$ in such a way that the condition $Fs$ is true in the final state?". When the answer is positive, the reasoning process returns a sequence of atomic actions that allows the achievement of the desired condition. The sequence is an execution trace of $p$ and can be seen as a *plan* to bring about $Fs$. This form of reasoning is known as *temporal projection*. Reasoning is done by exploiting a goal-directed proof procedure (denoted by "$\vdash$") designed for the language *DYnamics in LOGic* [3,4], which supports both temporal projection and planning and allows the proof of existential queries of kind (4). The definition of $p$ constrains the search space. For what concerns planning, the proof procedure allows the automatic extraction of linear or conditional plans for achieving the goal of interest from an incompletely specified initial state.

Let $\langle \mathcal{O}, \mathcal{P} \rangle$ be a service description. Let $p \in \mathcal{P}$ be a specific procedure and let us denote by $Q$ the query $Fs$ **after** $p$. Given a state $s_0$, containing all the fluents that are true in the beginning, we denote the fact that the query $Q$ is successful in the service description by $(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q$. The execution of the query returns as a side-effect an *execution trace* $\sigma$ of $p$, which is a sequence $a_1, \ldots, a_n$ of operations after which $Fs$ holds. We denote this by:

$$(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q \text{ w.a. } \sigma \qquad (5)$$

where "w.a." stands for *with answer*. The same reasoning can be applied to choreography roles. In this case, the answer will contain unbound operations.

A service can use temporal projection to decide whether it is possible to play a given role in a way that achieves a goal of interest.

**Example 2.3 Healthcare reservation (III part).** *Let us suppose that* $p1$ *is a service willing to play the* Patient *role. Suppose that its initial state is:*

$$s_0 = \left\{ \begin{array}{c} \mathbf{B}^{patient} examination, \\ \mathbf{B}^{patient} deferredPaymentPos, \\ \mathbf{B}^{patient} \neg listObtained, \\ \mathbf{B}^{patient} credentials \end{array} \right\}$$

*the truth value of all the other fluents is "unknown". In words,* $p1$ *knows the kind of medical examination to book, the needed credentials (i.e. the medical insurance id or the credit card number), the fact that it is possible to defer the payment directly to the medical center, and that no reservation process has started yet. The goal of* $p1$ *is to achieve the condition:*

$$Q = \{\mathbf{B}^{patient} reservationComplete, \\ \mathbf{B}^{patient} resNum\} \textbf{ after } \mathsf{bookExamination}$$

*Intuitively, the patient wants to have a reservation number after the interaction. By reasoning on the role and by using the definitions of the unbound operations that are given by the choreography,* $p1$ *can identify an execution trace, that leads to a state where* $Q$ *holds:*

$$\sigma = \mathsf{searchExamination}_{io}^{\gg}(examination, exmList); \\ \mathsf{askChosenExm}_{out}^{\ll}(chosenExm); \\ \mathsf{proceed}_{in}^{\ll}(); \\ \mathsf{choosePayment}_{io}^{\ll}(payMethods, chosenMethod); \\ \mathsf{doPayment}_{io}^{\gg}((credential, payInfo), resNum)$$

*This is possible because in a declarative representation specifications are executable. This execution does not influence the belief about the deferred payment, which persists from the initial through the final state and is not contradicted.*

## 3. Goal-driven selection

This section shows how to leverage the choreography role descriptions to the aim of selecting the services, that will play them, and the operations that will be invoked, in a way that preserves the achievement of the goals of interest. To this aim, we need to introduce the notion of flexible match (of a service operation to a choreography unbound operation), to discuss its impact on goal achievement, and to define the new notion of *conservative* substitution. We also report decidability results.

### *3.1. Matching unbound operations*

A choreography role composes all *unbound* operations. In order for a service to play it, it is necessary to bind them to service operations, which will either be supplied by the role player or by its interlocutors. Specifically, given a role, the operations invoked over the player, i.e. those that are decorated by the symbol $\ll$ ($operation^{\ll}$), must be bound to operations of the player. The others, that are decorated by the symbol $\gg$ ($operation^{\gg}$), must be bound to operations of its interlocutors. For instance, the Patient role of Figure 3 specifies operations, like *payment*, which are to be supplied by the role player, and others, like *searchExamination*, which are to be supplied by its interlocutor, i.e. the healthcare reservation system. The binding is possible only when all the partners in the interaction have been found. Notice that, in general, a service will have more than one operation matching with a single specification.

Let $\langle \mathcal{O}, \mathcal{P} \rangle$ be a role description, and let $\mathcal{O}_u^i \subseteq \mathcal{O}$ be the set of unbound operations that are to be supplied by the role player $S_i$. Let $\mathcal{O}_{S_i}$ be the set of operations that $S_i$ provides. In case $S_i$ is the player of the role $\langle \mathcal{O}, \mathcal{P} \rangle$, they will be operations decorated by $\ll$, otherwise they will be $\gg$ operations. We represent the binding of service operations to unbound operations by the substitution $\theta = [\mathcal{O}_{S_i}/\mathcal{O}_u^i]$[4] applied to $\langle \mathcal{O}, \mathcal{P} \rangle$. The application of the substitution is denoted by $\langle \mathcal{O}\theta, \mathcal{P}\theta \rangle$, where every element of $\mathcal{O}_u^i$ is substituted by/bound to an element of $\mathcal{O}_{S_i}$.

In the above definition, $\theta$ can be any kind of association between the operations of a service with the unbound operations of a choreography role. In practice it is the result of a *matching process*. In the literature it is possible to find many match algorithms, mostly based on the seminal work by Zaremski and Wing [46] on software components, and surveyed in [41]. Specifically, given a software component $I$, with precondition $I_{pre}$ and postcondition $I_{post}$, and a specification (or query, as it is called in the match-making community) $Q$, with precondition $Q_{pre}$ and postcondi-

[4]Notice that $[\mathcal{O}_{S_i}/\mathcal{O}_u^i]$ is actually a set of substitutions $[o/o_u]$ of single service operations to single unbound operations.

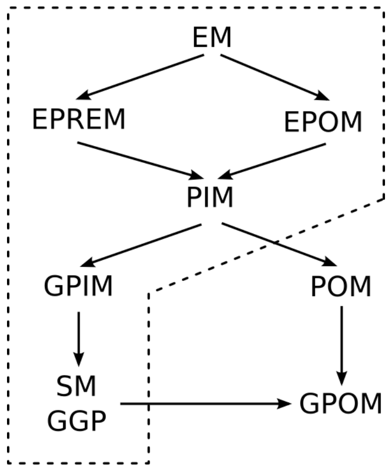

Fig. 4. Lattice of the local matches: on top the strongest. The dashed line highlights re-use ensuring matches. SM and GGP are in same node because logically equivalent.

tion $Q_{post}$, the most important kinds of relaxed match between $Q$ and $I$ are:

- EM (*Exact Pre/Post Match*): $Q_{pre} \Leftrightarrow I_{pre} \wedge Q_{post} \Leftrightarrow I_{post}$;
- EPREM (*Exact Pre Match*): $Q_{pre} \Leftrightarrow I_{pre} \wedge I_{post} \Rightarrow Q_{post}$;
- EPOM (*Exact Post Match*): $Q_{pre} \Rightarrow I_{pre} \wedge Q_{post} \Leftrightarrow I_{post}$;
- PIM (*Plugin Match*): $Q_{pre} \Rightarrow I_{pre} \wedge I_{post} \Rightarrow Q_{post}$;
- POM (*Plugin Post Match*): $I_{post} \Rightarrow Q_{post}$;
- GPIM (*Guarded Plugin Match*, a.k.a. Weak-Plugin [34]): $Q_{pre} \Rightarrow I_{pre} \wedge ((I_{pre} \wedge I_{post}) \Rightarrow Q_{post})$;
- SM (*Satisfies Match*, a.k.a. relaxed plug-in in [18], plug-in compatibility [21]): $Q_{pre} \Rightarrow S_{pre} \wedge (Q_{pre} \wedge I_{post} \Rightarrow Q_{post})$;
- GPOM (*Guarded Post Match*, a.k.a. Weak-Post [34]): $((I_{pre} \wedge I_{post}) \Rightarrow Q_{post})$;
- GGP (*Guarded-Generalized Predicate*): $(Q_{pre} \Rightarrow I_{pre}) \wedge ((I_{pre} \Rightarrow I_{post}) \Rightarrow (Q_{pre} \Rightarrow Q_{post}))$.

The different matches can be organized according to a lattice [41], that we reported in Figure 4. On top, there is the *Exact pre/post match*, which states the equivalence of $Q$ and $I$. Moving down in the lattice weaker and weaker match conditions are found. So, for the *Plugin match* it is sufficient that $I$ is behaviorally equivalent to $Q$ in the context where it replaces $Q$. The *Plugin post match* relaxes this requirement by considering only postconditions. *Guarded matches*, instead, focus on guaranteeing that the desired postcondition holds when the precondition of $I$ holds, not in general.

In our application domain, $Q$ is an *unbound operation*, while $I$ is a service *operation*. When needed, we decorate substitutions with an acronym denoting the applied match (e.g. $\theta_{EM}$ is a substitution obtained by applying the exact match).

Particularly interesting is the family of the so-called *re-use ensuring* matches. In the following, $M(I, Q)$ represents that a service $I$ matches with a specification $Q$ according to the specification match $M$.

**Definition 1 (Re-use ensuring match [18])** *A specification match $M$ is* re-use ensuring *iff for any $I$ and $Q$, $M(I, Q) \wedge \{I_{pre}\}I\{I_{post}\} \Rightarrow \{Q_{pre}\}I\{Q_{post}\}$*

In the above definition, $\{C_{pre}\}C\{C_{post}\}$ denotes a Hoare triple and is informally interpreted as the truth of "program $C$ started with $C_{pre}$ satisfied will terminate in a state such that $C_{post}$ holds" [26]. Notice that this is a local property, in fact, it only accounts for the specification of a single component. However, when a match is re-use ensuring, its execution in a state that contains the precondition expressed in the query produces the postcondition expressed in the query. Considering the lattice in Figure 4, re-use ensuring matches are all those inside the dashed line. Only POM and GPOM are not re-use ensuring because no assumption is made on their preconditions, which can be wider than those expressed by the query.

## 3.2. *Conservative re-use ensuring matches*

When the matching process is applied for deciding if a service could play a role in a (partially instantiated) choreography, the desire is that *the substitution preserves the properties of interest*, i.e. the goals that can be entailed by reasoning on each role description separately, should still be achievable after the substitution. We formalize this notion in the following way.

**Definition 2 (Conservative substitution)** *Consider a role $\langle \mathcal{O}, \mathcal{P} \rangle$ of a choreography, a query Q, and an initial state $s_0$ of a service, candidate to play the role, such that $(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q$ w.a. $\sigma$. A substitution $\theta$ is* conservative *when $(\langle \mathcal{O}\theta, \mathcal{P}\theta \rangle, s_0) \vdash Q$ w.a. $\sigma\theta$.*

All the matches described in the previous section (including also the re-use ensuring matches) were defined for the retrieval of single components, and have a *local* nature, i.e. they compare a single requirement (an unbound operation) to a single software specification (a service operation) independently of the *context of usage*. Therefore, substitutions produced by applying one of the described flexible matches generally are *not conservative*, i.e. goals that can be achieved when using the specification cannot be achieved anymore after the substitution. Intuitively, the reason is that besides a few special cases (EM and EPOM are the only matches which, by their own nature, are conservative), the identified operation can produce *additional effects* w.r.t. $Q_{post}$. When the operation is inserted in the context of a role execution, the additional effects may inhibit the preconditions of operations that follow. This is a problem because the choice of playing a role bases on the proof that the adoption of that role allows the achievement of a goal of interest for the player.

**Example 3.1 Healthcare reservation (IV part).** *The example shows how an implementation that does not exactly match with a specification may (or may not) prevent the applicability of other operations. Given the goal and service description specified in the previous parts of the example, suppose the unbound operation* choosePayment *of the role* Patient *is defined as:*

*(a)* $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$
$\quad$ **possible if** $\mathbf{B}^{patient}examination \wedge \mathbf{B}^{patient}exmSelected \wedge \mathbf{B}^{\mathbf{patient}}\mathbf{deferredPaymentPos}$

*(b)* $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$
$\quad$ **causes** $\mathbf{B}^{patient}payMethods$

*(c)* $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$
$\quad$ **causes** $\mathbf{B}^{patient}chosenMethod$

*(d)* $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$
$\quad$ **causes** $\mathbf{B}^{patient}sent(chosenMethod)$

*The precondition* $\mathbf{B}^{patient}$ deferredPaymentPos *amounts to the possibility of paying the service directly at the medical center. Now, consider a service, that could possibly play the role* Patient*. Suppose that it implements the operation* searchExamination *as follows, while all other operations exactly match with the corresponding specifications:*

*(a)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ **possible if** $\mathbf{B}^{patient}examination \wedge \mathbf{B}^{patient}\neg sellingStarted$

*(b)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ **causes** $\mathbf{B}^{patient}sent(examination)$

*(c)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ **causes** $\mathbf{B}^{patient}exmList$

*(d)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ **causes** $\mathbf{B}^{patient}listObtained$

*(e)* $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$
$\quad$ **causes** $\neg\mathbf{B}^{\mathbf{patient}}\mathbf{deferredPaymentPos}$

*This operation matches with the corresponding unbound operation according to many of the re-use ensuring matches (e.g. EPREM, PIM, GPIM, SM). However, it has an additional effect which negates the possibility of paying at the medical center (*$\neg\mathbf{B}^{patient}$ deferredPaymentPos*), and which* prevents the executability *of the operation* choosePayment*, as defined above.*

*If the additional effect of* searchExamination *were, instead,* $\mathbf{B}^{patient}$ resultsAtHome*, supplying an additional information concerning the possibility of sending results directly at the patient's home, the achievement of the goal would not be compromised.*

We now show how it is possible to enrich the *reuse ensuring* matches in a way that *guarantees the production of conservative substitutions*. We do so by exploiting only constraints that can be inferred from the choreography –which supplies the *global execution context*, in which unbound operations are immersed– without modifying the local nature of the matches.

The approach takes into account the *causal dependencies* between operations, where, intuitively, an operation depends on another when the latter produces some effect which is, then, used as a precondition by the former. The idea is to verify if the substitution breaks the "causal chain" which allows the execution of the sequence of operations of an execution $\sigma$, which originally brought to the achievement of a goal of interest. We use $\sigma$ for defining a set of constraints that, whenever satisfied by a substitution obtained by a reuse ensuring match, guarantee that the substitution is also conservative. This is a *sufficient* condition because there might exist conservative substitutions that do not satisfy this set of constraints.

Given a role (or a service) description $\langle \mathcal{O}, \mathcal{P} \rangle$, suppose that in the initial state $s_0$, the query $Q = Fs$ **after** $p$ succeeds with answer $\sigma = a_1; a_2; \ldots; a_n$. Let $\overline{\sigma}$ be the sequence of operations $a_0$; $a_1$; $a_2; \ldots$; $a_n$; $a_{n+1}$, i.e. $\sigma$ completed by adding two *fictitious* operations $a_0$ and $a_{n+1}$, that respectively represent the initial state $s_0$ and the goal $Fs$, which must hold after $\sigma$. Let us respectively denote by the functions $\mathsf{Precs}(a)$ and $\mathsf{Effs}(a)$ the preconditions and the effects of an action $a$: $\mathsf{Precs}(a_0) = \emptyset$, $\mathsf{Effs}(a_0) = s_0$, while $\mathsf{Effs}(a_{n+1}) = \emptyset$ and $\mathsf{Precs}(a_{n+1}) = Fs$.

**Definition 3 (Action dependency)** *Given a completed execution trace* $\overline{\sigma} = a_0; \ldots a_j; \ldots a_i; \ldots; a_{n+1}$*, where*

*$j < i$, we say that* in $\overline{\sigma}$ *the operation* $a_i$ depends on $a_j$ for the fluent $\mathbf{B}l$ *iff* $\mathbf{B}l \in \mathsf{Effs}(a_j)$, $\mathbf{B}l \in \mathsf{Precs}(a_i)$, *and $\neg\exists k$, $j < k < i$, such that $\mathbf{B}l \in \mathsf{Effs}(a_k)$. We denote this dependency by: $a_j \rightsquigarrow_{\langle \mathbf{B}l, \overline{\sigma} \rangle} a_i$.*

Intuitively, operation $a_i$ cannot be executed in the context of $\overline{\sigma}$ if the effects of $a_j$ do not hold. Algorithm 1 verifies the dependency of an action $a_i$ on an action $a_j$.

**Algorithm 1** Action dependency algorithm

```
Require: $j < i \wedge \mathbf{B}l \in \mathsf{Effs}(a_j) \wedge \mathbf{B}l \in \mathsf{Precs}(a_i)$
1: function ACTIONDEP($\overline{\sigma}$, $i$, $j$, $\mathbf{B}l$)
2:     for all $a_k \in a_{j+1}, \ldots, a_{i-1}$ do
3:         if $\mathbf{B}l \in \mathsf{Precs}(a_k)$ then
4:             return $false$
5:         end if
6:     end for
7:     return $true$
8: end function
```

**Definition 4 (Dependency set of a fluent)** *Given a fluent $\mathbf{B}l$ and a sequence of operations $\overline{\sigma}$, we define the* dependency set *of $\mathbf{B}l$ as* $\mathsf{Deps}(\mathbf{B}l, \overline{\sigma}) = \{(j, i) \mid a_j \rightsquigarrow_{\langle \mathbf{B}l, \overline{\sigma} \rangle} a_i\}$.

This notion captures the set of actions that, within an execution trace, cannot be executed if $\mathbf{B}l$ does not hold. Algorithm 2 computes the dependency set of a given $\mathbf{B}l$ within the completed execution trace $\overline{\sigma}$.

**Algorithm 2** Dependency set algorithm

```
1: function DEPS($\overline{\sigma}$, $\mathbf{B}l$)
2:     $ds \leftarrow \emptyset$
3:     for all $j \in 0 \ldots, n+1$ do
4:         for all $i \in 0 \ldots, n+1$ do
5:             if $actionDep(\overline{\sigma}, j, i, \mathbf{B}l)$ then
6:                 $ds \leftarrow ds \cup \{(j, i)\}$
7:             end if
8:         end for
9:     end for
10:    return $ds$
11: end function
```

**Definition 5 (Uninfluential fluent)** *Let $\mathbf{B}l$ be an additional effect of an operation $s$ that substitutes an unbound operation $o_u$ (i.e., $\mathbf{B}l \in \mathsf{Effs}(s) - \mathsf{Effs}(o_u)$), occurring at position $k$ of a completed execution trace $\overline{\sigma} = a_0; \ldots (o_u)_k; \ldots; a_{n+1}$. We say that $\mathbf{B}l$ is an* uninfluential fluent *w.r.t. the sequence $\overline{\sigma}$ iff for all pairs $(j, i) \in \mathsf{Deps}(\mathbf{B}\neg l, \overline{\sigma})$, we have that $k < j$ or $i \leq k$.*

Intuitively, this means that the fluent *will not break* any dependency between the operations which involve the inverse fluent because either it will be overwritten or it will appear after its inverse has already been used. When this holds for all additional fluents of a substitution, the substitution is said to be *uninfluential*.

**Example 3.2 Healthcare reservation (V part).** *In Example 3.1, the additional effect* $\mathsf{resultsAtHome}$ *would be uninfluential, while the additional effect* $\neg\mathbf{B}^{patient}$ $\mathsf{deferredPaymentPos}$ *would not.*

Algorithm 3 verifies if an additional fluent is uninfluential.

**Algorithm 3** Uninfluential fluent algorithm

```
Require: $\mathbf{B}l \in \mathsf{Effs}(s) - \mathsf{Effs}(o_u)$
1: function UNINFFLUENT($\overline{\sigma}$, $\mathbf{B}l$, $s$, $o_u$, $k$)
2:     for all $(j, i) \in deps(\overline{\sigma}, \mathbf{B}l)$ do
3:         if $\neg(k < j \vee i \leq k)$ then
4:             return $false$
5:         end if
6:     end for
7:     return $true$
8: end function
```

**Definition 6 (Uninfluential substitution)** *A substitution $\theta$ is called* uninfluential *iff for all operation substitutions $[s/o_u]$ in $\theta$, all beliefs in* $\mathsf{Effs}(s) - \mathsf{Effs}(o_u)$ *are uninfluential fluents w.r.t. $\overline{\sigma}$.*

Algorithm 4 verifies if a substituion is uninfluential.

**Algorithm 4** Uninfluential substitution algorithm

```
1: function UNINFSUB($\overline{\sigma}$, $\theta$)
2:     for all $[s/o_u] \in \theta$ do
3:         for all $\mathbf{B}l \in \mathsf{Effs}(s) - \mathsf{Effs}(o_u)$ do
4:             $k \leftarrow$ position of $o_u$ in $\overline{\sigma}$
5:             if $\neg uninfFluent(\overline{\sigma}, \mathbf{B}l, s, o_u, k)$ then
6:                 return $false$
7:             end if
8:         end for
9:     end for
10:    return $true$
11: end function
```

**Theorem 3.1** *The problem of determining whether a substitution is uninfluential w.r.t. an execution trace is* decidable.

**Proof 3.1** *All the reported algorithms rely on nested* for-loops*, thereby they always end.*

Finally, the following theorem relates the notion of *uninfluential substitution* and the notion of *conservative substitution*, showing that uninfluential substitutions are also be *conservative*.

**Theorem 3.2** *Let $M$ be a re-use ensuring match, any substitution $\theta_M$ that is uninfluential is also conservative.*

**Proof 3.2** *The proof is by absurd and it uses the proof theory introduced in [3]. Let us assume that $(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q$ w.a. $\sigma = a_0, a_1, \ldots, a_{i-1}, a_i, \ldots, a_n$ but $(\langle \mathcal{O}\theta_M, \mathcal{P}\theta_M \rangle, s_0) \nvdash Q$ w.a. $\sigma\theta_M$.*

*This means that there exists a fluent $F$ such that $a_0, a_1, \ldots, a_{i-1} \vdash F$ but $(a_0, a_1, \ldots, a_{i-1})\theta_M \nvdash F$, where $F \in \mathsf{Precs}(a_i)$, i.e. $a_i$ is not executable because one of its preconditions does not hold.*

*Now, since $a_0, a_1, \ldots, a_{i-1} \vdash F$, there exists $j \leq i-1$, such that $a_0, a_1, \ldots, a_j \vdash F$ and $F \in \mathsf{Effs}(a_j)$ but $(a_0, a_1, \ldots, a_j)\theta_M \nvdash F$. Let us assume that $j$ is the last operation to produce $F$ before $a_i$. There are two possible cases, either $F \notin \mathsf{Effs}(a_j\theta_M)$ or there is another operation $a_k\theta_M$, with $j < k < i$, such that $\neg F$ is one of its effects.*

*The first case is absurd because, by hypothesis, the match is re-use ensuring, therefore $(a_0, a_1, \ldots, a_{i-1}, a_i)\theta_M \vdash F$, for any fluent $F$ in $\mathsf{Effs}(a_i)$. The second is absurd as well, since $j$ is the last operation to produce $F$, the effect $\neg F$ of $a_k\theta_M$ should be one of its additional effects but this is absurd because by hypothesis $\theta_M$ is an uninfluential substitution.*

Theorem 3.2 provides a *sufficient condition* for proving that a substitution is conservative. Indeed, an operation may have additional effects that break some dependency which are compensated by additional effects of other operations.

### *3.3. Remarks*

The choice of not considering more complex forms of dependencies is motivated by the fact that in this way the substitution of a single operation *can be kept local* because it does not depend on other substitutions. The important consequence is that *conservative substitutions can be built by applying a linear, feedforward process without the need of backtracking*. In fact, it is sufficient to verify for each $[s/o_u]$ that the additional fluents introduced by $s$ with respect to $o_u$ are uninfluential, and this can be done simply by considering the dependency sets of the unbound operations that follow $s$ along the considered $\sigma$. This is a fundamental property that allows operations to be executed as soon as they are selected with the guarantee that it will not be necessary to perform any backtracking – which would require the rollback of all the operations executed up to that point.

The framework can also be used to allow a service to select a possible partner. Let us show this with an example. Consider a choreography with two roles $\mathcal{C} = (R_1, R_2)$ and a service $S_1$ that wishes to play role $R_1$ with the aim of achieving a personal goal $Fs_1$ starting from the initial state $s_0$. Assume that $p_1$ is the top level procedure of $R_1$ and that $S_1$ verified that $\langle R_1, s_0 \rangle \vdash Fs_1$ **after** $p_1$ w.a. $\sigma$. Now consider a service $S_2$ that ideally could play the role $R_2$ and that published the interface of the operations that it provides. $S_1$ can use this description to start a matching procedure, aimed at identifying a substitution $\theta^{R_2}$, which provides the necessary bindings of operations provided by $S_2$ to unbound operations that are up to $R_2$ in $p_1$. Moreover, $S_1$ can verify if substitutions $\theta^{R_1}$ and $\theta^{R_2}$ are uninfluential w.r.t. $\sigma$ (see Definition 6) and therefore also conservative (by Theorem 3.2), i.e. if $\langle R_1\theta^{R_1}\theta^{R_2}, s_0 \rangle \vdash Fs_1$ **after** $p_1\theta^{R_1}\theta^{R_2}$ w.a. $\sigma\theta^{R_1}\theta^{R_2}$. When this happens, $S_1$ has found a partner to interact with in a way that respects the choreography and that allows it to achieve its goal.

## 4. Joint achievement of individual goals

So far, we have motivated the participation of a service to a choreography, saying that the service will take on a role only if by doing so it will have the possibility of satisfying its purposes. The issue that we discuss in this section is whether it is possible for a team of services, which would play different roles, to reach an agreement about their possible executions so that all of them will achieve their individual goals. To have an intuition of the problem, consider a choreography $\mathcal{C} = (R_1, R_2)$. Consider also two services, $S_1$ and $S_2$, with goals $G_1$ and $G_2$, which are interested in playing respectively $R_1$ and $R_2$. Suppose that the first service identifies the execution trace $\sigma_1$ of $R_1$ as allowing it to achieve its goal, while the second service identifies the execution trace $\sigma_2$ of $R_2$. Intuitively, the issue is that $\sigma_1$ and $\sigma_2$ might not be complementary views of a same execution, so $\sigma_1$ may include the invocation of $operation^{\gg}$ when $\sigma_2$ does not include the corresponding execution of $operation^{\ll}$. This is a limit case. When a party identifies not just a single execution but a set of alternatives $\sigma_i$, part of them may be compatible with some of the executions traces identified by the other agents. The problem, in this case, is to converge to a common solution, that is to identify the subset of the identified executions which allows the achievement of all goals.

The coordination of a set of autonomous, cooperating parties is a well-known issue in multi-agent systems research [44]. Approaches include, among others, coordination as a *post-planning process* [39], where constraints are checked after the plan was found, and the use of *conversation moderators* [38], that guarantee that the shared objectives will be reached. In general, the reasoning applied in these cases has a local nature because the parties do not have complete knowledge about one another. On the contrary, approaches which rely on classical game theory [31], assume complete knowledge about all of the parties: a quite unrealistic assumption, especially if one considers applications. The approach that we outline implements a simple form of negotiation, inspired by [23,24]. Let us explain it, focussing on two-role choreographies. To this aim, we introduce a few useful notions.

Given an operation $a$, we denote by $\overline{a}$ its *complementary operation*. For instance, in Example 2.1 $\mathsf{searchReservation}^{\gg}_{rr}$ is complementary to $\mathsf{searchReservation}^{\ll}_{rr}$. This notion can easily be extended to execution traces:

**Definition 7 (Compatible execution traces)** *Let $\sigma = a_0; \dots ; a_n$ and $\sigma' = a'_0; \dots ; a'_n$ be two execution traces, we say that $\sigma$ is* compatible *with $\sigma'$ iff for each operation $a_i$ in $\sigma$ the corresponding $a'_i$ in $\sigma'$ is its complementary view.*

Let us consider a two-role choreography and see how two services $S_1$ and $S_2$ can identify a set of compatible traces, whose execution allows the achievement of both their personal goals. Suppose (i) that $S_1$ has identified the execution trace $\sigma$ of the top procedure $p_1$ of $R_1$, that allows the achievement of its goal $Fs_1$, (ii) that it can implement the operations that should be supplied by the player of $R_1$ by using the substitution $\theta^{R_1}$, (iii) that it has identified in $S_2$ a possible partner and in $\theta^{R_2}$ a possible substituion for the operations that are to be supplied by the player of $R_2$, and (iv) that $S_1$ verified that the substitutions $\theta^{R_1}$ and $\theta^{R_2}$ are conservative[5]. As a result, $S_1$ now has an execution trace $\sigma\theta^{R_1}\theta^{R_2}$ that does not contain unbound operations.

[5]The two substitutions involve disjoint sets of unbound operations by construction.

Before executing it, its candidate partner $S_2$ must agree on executing the complementary trace $\overline{\sigma\theta^{R_1}\theta^{R_2}}$. Of course, $S_2$ might have its own goal $Fs_2$ which should be achieved with the execution of the top level procedure of the role $R_2$, that we here call $p_2$. Therefore, the following conditions must be verified:

1. $\overline{\sigma\theta^{R_1}\theta^{R_2}}$ must be an execution trace of $p_2$;
2. after executing $\overline{\sigma\theta^{R_1}\theta^{R_2}}$ the goal $Fs_2$ must hold.

The first condition is guaranteed by the assumption that the choreography is well-defined, i.e., that its roles are by construction *interoperable* and, therefore, for any execution trace that a role can enact, the other role supplies a complementary trace[6]

The second condition is to be verified by $S_2$ by checking if $\langle R_2\overline{\theta^{R_1}\theta^{R_2}}, s^2_0\rangle \vdash Fs_2$ **after** $\overline{\sigma\theta^{R_1}\theta^{R_2}}$, where $s^2_0$ is the initial state of $S_2$ and $\overline{\theta^{R_1}\theta^{R_2}}$ is the substitution obtained from $\theta^{R_1}\theta^{R_2}$ by using the complementary views of the involved operations. If the answer is positive, the two partners will have identified an execution that fits both their goals. Notice that the reasoning applied by $S_2$ is much simpler than the one executed on $p_1$ because $S_2$ only has to check if the goal is satisfied, while the reasoning applied to $p_1$ was aimed at identifying such a trace.

More in general, $S_1$ will identify a set $\Sigma$ of alternative execution traces, each of which allows the achievement of $Fs_1$, and then propose $\Sigma$ to $S_2$. $S_2$ will restrict them to a set $\Sigma' \subseteq \Sigma$, whose elements satisfy the goals of both parties. Notice that *none of the services* is requested to disclose its goal to its partner.

This interaction procedure, that we have described informally, allows the joint achievement of goals and can be seen as a kind of *negotiation* [31,44]. If no other preference criterion is specified, it is *not important* which execution trace in $\Sigma'$ will be enacted, and it is not necessary to perform any further negotiation step because both the choreography and $\Sigma'$ are known to the partners: whoever the initiator of the interaction will be, each partner knows how to behave. Each partner also has the means for understanding when the interaction takes an unagreed path and decide accordingly, for instance by stopping the interaction.

It is worth noting that, in general, the set of alternative execution traces might not be finite. Moreover, the derivation ($\vdash$) is semidecidable unless the choreographies are properly restricted, for instance by focussing

[6]Discussing interoperability is out of the scope of this work but the interested reader can check [36,13,15,7].

onto regular sets [3]. Indeed, many choreographies are of this kind [22]. As a final remark, the fact that the two services converged to a set of promising execution traces does not imply that, at execution time, they will be able to stick to them. The reason is that those traces were identified by making assumptions on the values returned by tests and conditions, which cannot be known in advance. The actual results of such tests, obtained at execution time, could be different than the assumed ones.

## 5. Conclusions and Related Work

This article surveys and extends the proposal in [6,8], a work that concerns the use of rule-based choreography specifications in the process of performing a goal-driven selection of services which should act as role players. This activity is quite complex and involves many steps, including the verification of the possible achievement of individual goals when playing roles, the matchmaking of operations against specifications contained in the choreography, and the identification of a joint working plan, that allows all the participants to pursue their individual goals collectively.

For what concerns *matchmaking*, we show that for any re-use ensuring match, as defined in [18], it is decidable to verify if a substitution is uninfluential and, therefore, conservative w.r.t. a goal, which can be satisfied when using the unbound operations given by a choreography. This result allows the enrichment of the matches with a test that can be applied *locally*, i.e. operation by operation, while the search of all the necessary operations progresses along the examined execution trace. The test guarantees that any goal that can be achieved by reasoning on a role specification will still be achievable after the role players have been selected. This approach is made possible by the availability of the choreography, which supplies the *execution context* of the various operations.

The literature on matchmaking is wide and it is really difficult to be exhaustive. Most of the matchmaking techniques that were proposed concern the selection of a service (or a service operation) as if it had to be used alone. The matches proposed by Zaremski and Wing in [46] inspired most of the semantic matches for web service discovery. Amongst them, Paolucci et al. [33] propose four degrees of match (exact, plugin, subsumes, and fail). Differently than in our proposal, these matches tackle DAML-S representations, in which services are described by means of *inputs* and *outputs*; specifically, matches are computed on the ontological relations of the outputs of a service advertisement and a query. This approach is refined in [29], which describes a service matchmaking prototype, which uses a Description Logic reasoner to compare ontology-based service descriptions. As in [33], the matchmaking process produces a discrete scale of degrees of match. Approaches along this line are orthogonal to our work. The framework WSMO [20], instead, does not suggest a specific matching rule, which is up to the specific implementations, although in [28] the authors propose an approach that, as well as the previous ones, relies on [46,29].

Recently some approaches that take into account also the *execution context* were proposed [25,40]. Our proposal can be placed in this category but differs from the cited works in many respects. The approach in [25] is inserted in a top-down specification procedure which starts from the specification of a business template (something analogous to a choreography) and ends with the retrieval of the needed services. The perspective is the one of the designer. The issue of whether deciding to play a role is not posed, and consequently no notion of goal is introduced. The approach is "contextual" in that it uses the abstract specification of the business template for producing the desired service specifications: using our terminology, it builds a choreography including a characterization of unbound operations. On the other hand, [40] faces a problem that is complementary to the one we faced, because it supplies a formal setting, based on colored Petri nets, for building mediators between services that must interact but whose interfaces and message flows do not precisely match. The two services are seen as immersed in a BPEL specification.

The approach we propose, for performing goal-driven service selection, is based on techniques for reasoning about actions and change, in a framework where choreography roles and service policies are modeled as procedures based on atomic operations. In particular, the technical framework enables the use of *procedural planning* techniques [4]. Planning techniques have been successfully used in the web service domain for building goal-driven adaptable compositions [12,27,16,35] but in ways that differ from our proposal. For instance, in [12,35] planning techniques are used for producing service orchestrations, starting from a set of process-level service descriptions, without reference to any choreography. In our proposal, instead, the specification of an abstract choreography already provides a composition pattern to refer to, and drives the service selection process: the joint use of reasoning about actions and of semantic matchmaking plays a crucial role in the definition of flexible selections, adapted to given sets of requirements (the goals). Within the community of researchers who study the service composition problem by using planning, the value of semantic matchmaking is highlighted in [35,16], where ontology-based service descriptions are functional to the definition of local matches. Our work does exploit semantic descriptions, but with the significant difference of scaling up from a local to a global perspective, once again thanks to the availability of choreographies.

For what concerns the *joint achievement of goals*, the discussion in Section 4 is inspired by [23,24] but there are some important differences that characterize and distinguish the two proposals. The first is that in our proposal the behavior of services is ruled by a choreography, which specifies all the possible interactions. When services play choreography roles, they commit to keep their behavior adherent to the specification. The proposal by Ghaderi et al., instead, does not rely on choregraphies (nor on any other kind of interaction specifications). The reasoning process considers all the possible executions that can be composed out of a set of atomic actions. Due to the lack of a choreography, when an execution trace, that allows the joint achievement of goals, is identified, there is still the need of adding a coordination mechanism that enables its actual execution. This step is not necessary in our case.

Another difference is that in [23,24] each agent reasons also for its partner, because the two share a common goal and a common state. The identified joint plans correspond to the agents preferred strategies and are obtained by the iterative elimination of dominated strategies. In our framework, partners does not have any knowledge about the others' goals, as it is reasonable to suppose for web services. Therefore, the execution traces that we identify are not necessarily dominant. For example, a greedy partner, at some point of its execution, may take an action (if the protocol includes it) that is not in the agreement but that allows the immediate achievement of its own goal. This behavior is, however, not convenient over the long run because since its interlocutor knows the choreography and knows the agreed traces, it has a way for *monitoring* the on-going course of interaction. As soon as it realizes that there is a deviation from the agreed traces, it can take an appropriate action, e.g. interrupt the interaction or compensate certain executed actions, or even publish a low reputation rate for the partner. This situation corresponds, in game theory, to a game iterated forever, i.e. a game in which each round is followed by another one [44].

Another interesting issue concerns preference criteria that services may apply in order to *rank strategies*. Supposing that the two services decide to behave well and to execute a trace which is in the agreement, how can they converge to a best-compromise agreement? One criterion could be to select the trace which entails the minimum number of effects. For example, one can imagine that a patient prefers a trace at the end of which it has simply booked the desired medical examination w.r.t. one in which it has additionally been registered in the advertisement mailing list of the medical centre. We mean to investigate this issue as part of future works.

## References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, 2004.

[2] M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension*. PhD thesis, Dipartimento di Informatica, Università degli Studi di Torino, Italy, 1998.

[3] M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, 41(2-4):207–257, 2004.

[4] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *JLAP, special issue on Web Services and Formal Methods*, 70(1):53–73, 2007.

[5] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Reasoning on choreographies and capability requirements. *International Journal of Business Process Integration and Management*, 2(4):247–261, 2007.

[6] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Service selection by choreography-driven matching. In T. Gschwind and C. Pautasso, editors, *Emerging Web Services Technology*, volume II of *Whitestein Series in Software Agent Technologies and Autonomic Computing*, chapter 1, pages 5–22. Birkhäuser, September 2008.

[7] M. Baldoni, C. Baroglio, A.K. Chopra, N. Desai, V. Patti, and M.P. Singh. Choice, interoperability, and conformance in interaction protocols and service choreographies. In *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

[8] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Goal preservation by choreography-driven matchmaking. In E. Di Nitto and M. Ripeanu, editors, *Service-Oriented Computing - ICSOC 2007 Workshops, Revised Selected Papers*, volume 4907 of *LNCS*, pages 413–426. Springer, 2009.

[9] U. Bellur, H. Vadodaria, and A. Gupta. *Greedy Algorithms*, chapter Semantic Matchmaking Algorithms, pages 481–502. InTech, 2008.

[10] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Synthesis of Underspecified Composite e-Service bases on Atomated Reasoning. In *Proc. of ICSOC04*, pages 105–114. ACM, 2004.

[11] F. Bergenti and A. Poggi. Developing smart emergency applications with multi-agent systems. *IJEHMC*, 1(4):1–13, 2010.

[12] P. Bertoli, M. Pistore, and P. Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3-4):316 – 361, 2010.

[13] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are two web services compatible? In *TES 2004*, volume 3324 of *LNCS*, pages 15–28. Springer, 2005.

[14] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta. CooWS: Adaptive BDI agents meet service-oriented computing. In *Proceedings of the Int. Conference on WWW/Internet*, pages 205–209, 2005.

[15] M. Bravetti and G. Zavattaro. Contract based multi-party service composition. In *FSEN*, volume 4767 of *LNCS*, pages 207–222. Springer, 2007.

[16] J. Bryson, D. Martin, S. McIlraith, and L. A. Stein. Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web. In *Web Intelligence*. Springer, 2003.

[17] G. Caire, D. Gotta, and M. Banzi. Wade: a software platform to develop mission critical applications exploiting agents and workflows. In M. Berger, B. Burg, and S. Nishiyama, editors, *AAMAS (Industry Track)*, pages 29–36, 2008.

[18] Y. Chen and B. H. C. Cheng. *Foundations of Component-Based Systems*, chapter A Semantic Foundation for Specification Matching, pages 91–109. Cambridge Univ. Press, 2000.

[19] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Reasoning about agents and protocols via goals and commitments. In *AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 457–464, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-98265-710-0, 978-0-9826571-1-9.

[20] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres. *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer, 2007.

[21] B. Fischer and G. Snelting. Reuse by contract. In *ESEC/FSE-Workshop on Foundations of Component-Based Systems*, 1997.

[22] Foundation for Intelligent Physical Agents. http://www.fipa.org.

[23] H. Ghaderi, H. Levesque, and Y. Lespérance. A logical theory of coordination and joint ability. In *Proc. of AAAI'07*, pages 421–426, 2007.

[24] H. Ghaderi, H. Levesque, and Y. Lespérance. Towards a logical theory of coordination and joint ability. In *Proc. of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07)*, pages 532–534, 2007.

[25] Weili Han, Xingdong Shi, and Ronghua Chen. Process-context aware matchmaking for web service composition. *J. Netw. Comput. Appl.*, 31:559–576, November 2008.

[26] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[27] Eirini Kaldeli, Alexander Lazovik, and Marco Aiello. Continual planning with sensing for web service composition. *AAAI Conference on Artificial Intelligence*, 2011.

[28] U. Keller, R. Laraand A. Polleres, I. Toma, M. Kifer, and D. Fensel. D5.1 v0.1 wsmo web service discovery. Technical report, WSML deliverable, 2004.

[29] L. Li and I. Horrocks. A software framework for matchmaking based on semantic technology. In *Proc. of WWW Conference*. ACM Press, 2003.

[30] OASIS. Web services business process execution language version 2.0, working draft. URL `http://www.oasis-open.org/`.

[31] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, July 1994.

[32] OWL-S Coalition. URL `http://www.daml.org/services/owl-s/`.

[33] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *Proc. of ISWC '02*, pages 333–347. Springer, 2002.

[34] J. Penix and P. Alexander. Efficient specification-based component retrieval. *Automated Software Engg.*, 6(2):139–170, 1999.

[35] Marco Pistore, Luca Spalazzi, and Paolo Traverso. A minimalist approach to semantic annotations for web processes compositions. In York Sure and John Domingue, editors, *ESWC*, volume 4011 of *LNCS*, pages 620–634. Springer, 2006.

[36] S. K. Rajamani and J. Rehof. Conformance checking for models of asynchronous message passing software. In *CAV*, volume 2404 of *LNCS*, pages 166–179. Springer, 2002.

[37] A. Ricci, M. Piunti, and M. Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, pages 1–35, 2010. ISSN 1387-2532.

[38] C. Sibertin-Blanc and N. Hameurlain. Participation Components for Holding Roles in Multiagent Systems Protocols. In *Engineering Societies in the Agents World*, volume 3451 of *Lecture Notes in Computer Science*, pages 60–73, August 2005.

[39] J. Renze Steenhuisen, Cees Witteveen, Adriaan ter Mors, and Jeroen Valk. Framework and Complexity Results for Coordinating Non-cooperative Planning Agents. In *MATES*, volume 4196 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2006.

[40] Wei Tan, Yushun Fan, and MengChu Zhou. A petri net-based method for compatibility analysis and composition of web services in business process execution language. *Automation Science and Engineering, IEEE Transactions on*, 6(1):94 –106, jan. 2009.

[41] Herbert Toth. On theory and practice of assertion based software development. *Journal of Object Technology*, 4(2):109–129, 2005.

[42] M. B. van Riemsdijk and M. Wirsing. Comparing goal-oriented and procedural service orchestration. *Multiagent and Grid Systems*, 6(2):133–163, 2010.

[43] Pengwei Wang, Zhijun Ding, Changjun Jiang, and Mengchu Zhou. Web service composition techniques in a health care service platform. In *ICWS*, pages 355–362. IEEE Computer Society, 2011. ISBN 978-1-4577-0842-8.

[44] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, March 2002.

[45] WS-CDL. URL `http://www.w3.org/TR/ws-cdl-10/`.

[46] A. Moormann Zaremski and J. M. Wing. Specification matching of software components. *ACM Transactions on SEM*, 6(4): 333–369, 1997.

## 6. Medical examination reservation example

For the sake of completeness, this appendix reports in details all operations of the running example.

Operation *searchExamination*, patient's view:

(a) $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$ **possible if** $\mathbf{B}^{patient}examination \wedge \mathbf{B}^{patient}\neg listObtained$
(b) $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$ **causes** $\mathbf{B}^{patient}sent(examination)$
(c) $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$ **causes** $\mathbf{B}^{patient}exmList$
(d) $\mathsf{searchExamination}^{\gg}_{io}(examination, exmList)$ **causes** $\mathbf{B}^{patient}listObtained$

Operation *searchExamination*, healthcare reservation service's view:

(a) $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$ **possible if** $true$
(b) $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$ **causes** $\mathbf{B}^{hrr}examination$
(c) $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$ **causes** $\mathbf{B}^{hrr}exmList$
(d) $\mathsf{searchExamination}^{\ll}_{io}(examination, exmList)$ **causes** $\mathbf{B}^{hrr}sent(exmList)$

Operation *askChosenExm*, patient's view:

(a) $\mathsf{askChosenExm}^{\ll}_{out}(chosenExm)$ **possible if** $\mathbf{B}^{patient}exmList$
(b) $\mathsf{askChosenExm}^{\ll}_{out}(chosenExm)$ **causes** $\mathbf{B}^{patient}chosenExm$
(c) $\mathsf{askChosenExm}^{\ll}_{out}(chosenExm)$ **causes** $\mathbf{B}^{patient}sent(chosenExm)$

Operation *askChosenExm*, healthcare reservation service's view:

(a) $\mathsf{askChosenExm}^{\gg}_{out}(chosenExm)$ **possible if** $\mathbf{B}^{hrr}exmList$
(b) $\mathsf{askChosenExm}^{\gg}_{out}(chosenExm)$ **causes** $\mathbf{B}^{hrr}chosenExm$

Operation *noBusiness*, patient's view:

(a) $\mathsf{noBusiness}^{\ll}_{in}()$ **possible if** $true$
(b) $\mathsf{noBusiness}^{\ll}_{in}()$ **causes** $\mathbf{B}^{patient}reservationAborted$

Operation *noBusiness*, healthcare reservation service's view:

(a) $\mathsf{noBusiness}^{\gg}_{in}()$ **possible if** $\mathbf{B}^{hrr}chosenOffer$
(b) $\mathsf{noBusiness}^{\gg}_{in}()$ **causes** $\mathbf{B}^{hrr}reservationAborted$

Operation *proceed*, patient's view:

(a) $\mathsf{proceed}^{\ll}_{in}()$ **possible if** $true$
(b) $\mathsf{proceed}^{\ll}_{in}()$ **causes** $\mathbf{B}^{patient}examinationSelected$

Operation *proceed*, healthcare reservation service's view:

(a) $\mathsf{proceed}^{\gg}_{in}()$ **possible if** $\mathbf{B}^{hrr}offer$
(b) $\mathsf{proceed}^{\gg}_{in}()$ **causes** $\mathbf{B}^{hrr}examinationSelected$

Operation *proposeExamination*, patient's view:

(a) $\mathsf{proposeExamination}^{\gg}_{io}((medicalCenter, examination), exmList)$ **possible if** $\mathbf{B}^{patient}medicalCenter \wedge \mathbf{B}^{patient}examination \wedge \mathbf{B}^{patient}\neg listObtained$
(b) $\mathsf{proposeExamination}^{\gg}_{io}((medicalCenter, examination), exmList)$ **causes** $\mathbf{B}^{patient}sent(medicalCenter) \wedge \mathbf{B}^{patient}sent(examination)$
(c) $\mathsf{proposeExamination}^{\gg}_{io}((medicalCenter, examination), exmList)$ **causes** $\mathbf{B}^{patient}exmList$
(d) $\mathsf{proposeExamination}^{\gg}_{io}((medicalCenter, examination), exmList)$ **causes** $\mathbf{B^{patient}listObtained}$

Operation *proposeExamination*, healthcare reservation service's view:

(a) $\mathsf{proposeExamination}^{\ll}_{io}((medicalCenter, examination), exmList)$ **possible if** $true$
(b) $\mathsf{proposeExamination}^{\ll}_{io}((medicalCenter, examination), exmList)$ **causes** $\mathbf{B}^{hrr}medicalCenter \wedge \mathbf{B}^{hrr}examination$
(c) $\mathsf{proposeExamination}^{\ll}_{io}((medicalCenter, examination), exmList)$ **causes** $\mathbf{B}^{hrr}exmList$
(d) $\mathsf{proposeExamination}^{\ll}_{io}((medicalCenter, examination), exmList)$ **causes** $\mathbf{B}^{hrr}sent(exmList)$

Operation *choosePayment*, patient's view:

(a) $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$ **possible if** $\mathbf{B}^{patient}chosenExm \wedge \mathbf{B}^{patient}examinationSelected \wedge \mathbf{B}^{patient}deferredPaymentPos$
(b) $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$ **causes** $\mathbf{B}^{patient}payMethods$
(c) $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$ **causes** $\mathbf{B}^{patient}chosenMethod$
(d) $\mathsf{choosePayment}^{\ll}_{io}(payMethods, chosenMethod)$ **causes** $\mathbf{B}^{patient}sent(chosenMethod)$

Operation *choosePayment*, healthcare reservation service's view:

(a) $\mathsf{choosePayment}^{\gg}_{io}(payMethods, chosenMethod)$ **possible if** $\mathbf{B}^{hrr}payMethods \wedge \mathbf{B}^{hrr}examinationSelected$
(b) $\mathsf{choosePayment}^{\gg}_{io}(payMethods, chosenMethod)$ **causes** $\mathbf{B}^{hrr}sent(payMethods)$
(c) $\mathsf{choosePayment}^{\gg}_{io}(payMethods, chosenMethod)$ **causes** $\mathbf{B}^{hrr}chosenMethod$

Operation *doPayment*, patient's view:

(a) $\mathsf{doPayment}^{\gg}_{io}((credential, payInfo), resNum)$
**possible if** $\mathbf{B}^{patient} credentials \wedge$
$\mathbf{B}^{patient} chosenExm \wedge \mathbf{B}^{patient} chosenMethod$

(b) $\mathsf{doPayment}^{\gg}_{io}((credential, payInfo), resNum)$
**causes** $\mathbf{B}^{patient} resNum$

(c) $\mathsf{doPayment}^{\gg}_{io}((credential, payInfo), resNum)$
**causes** $\mathbf{B}^{patient} reservationComplete$

Operation *doPayment*, healthcare reservation service's view:

(a) $\mathsf{doPayment}^{\ll}_{io}((credential, payInfo), resNum)$
**possible if** $\mathbf{B}^{hrr} chosenExm \wedge \mathbf{B}^{hrr} chosenMethod$

(b) $\mathsf{doPayment}^{\ll}_{io}((credential, payInfo), resNum)$
**causes** $\mathbf{B}^{hrr} credentials$

(c) $\mathsf{doPayment}^{\ll}_{io}((credential, payInfo), resNum)$
**causes** $\mathbf{B}^{hrr} resNum$

(d) $\mathsf{doPayment}^{\ll}_{io}((credential, payInfo), resNum)$
**causes** $\mathbf{B}^{hrr} sent(resNum)$

(e) $\mathsf{doPayment}^{\ll}_{io}((credential, payInfo), resNum)$
**causes** $\mathbf{B}^{hrr} reservationComplete$