Conservative re-use ensuring matches for service selection

Matteo Baldoni, Cristina Baroglio, Viviana Patti, and Claudio Schifanella
Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
{baldoni,baroglio,patti,schi}@di.unito.it

***Abstract*—The greater and greater quantity of services that are available over the web causes a growing attention to techniques that facilitate their reuse. A web service specification can be quite complex, including various operations and message exchange patterns. In this work, we give a declarative representation of services, and in particular of WSDL operations, that enables the application of techniques for reasoning about actions and change. By means of these techniques it becomes possible to reason on the specification of choreography roles and on possible role players, as a basis for selecting services which match in a flexible way with the specifications. Flexible match is, indeed, fundamental in order to enable web service reuse but it does not guarantee the preservation of the goals, that can be proved over the role specification itself. We show how to enrich various kinds of match proposed in the literature so to produce substitutions that preserve goals.**

I. Introduction

One of the key ideas behind web services is that services should be amenable to automatic retrieval, in order to allow the direct invocation as well as the composition with other services in order to fulfill a query. Nowadays, however, retrieval cannot yet be accomplished automatically as well and precisely as desired because the representations used for web services and the discovery mechanisms are semantically poor. The need of adding a *semantic layer* to service descriptions brought to initiatives like the development of OWL-S [1] and the development of the Web Service Modeling Ontology (WSMO) [2]. In these approach a richer annotation, aimed at representing the so called IOPEs (inputs, outputs, preconditions and effects of the service), is used. Inputs and outputs are usually described in terms taken from a public ontology, while preconditions and effects are often expressed by means of logic representations. A similar representation, based on preconditions and postconditions, is also typical of *design by contract*, originally introduced by Meyer for the Eiffel^{TM} language [3]. Here preconditions are the part of the contract which is to be guaranteed by the client; if this condition is guaranteed in the execution context of a method, then the server commits to guaranteeing that the postcondition holds in the state reached by the execution.

Semantic annotation allows the discovery of services, whose descriptions *do not exactly match* with the corresponding queries (e.g. [4], [5], [2]). Semantic matchmaking focuses on the discovery of single services, in the sense that a service is considered as corresponding to a *single operation*. In general, however, the use of a web service implies the execution of a *sequence of operations* in a particular *order*, which might even involve other services [6]: for instance, the clients of a supplier web service have to identify themselves, request item prices and delivery time, and so on. In order for the interaction to be successful, the interaction must obey some constraints: if they are not satisfied the service will be unable to proceed and will return an error. To allow the interaction, web services exhibit *interfaces* (port-types) which gather various operations that are logically related.

On the other hand, the need of describing *compositions* of services, which have to interact according to (complex) patterns of interaction, ruled by conversation protocols, has lead to the development of choreography languages like WS-CDL [7]. WS-CDL is aimed at describing collaborations between any type of participant independently from the programming model used by its implementation. Also a WS-CDL specification can be seen as a sort of contract, that specifies the ordering conditions and constraints that rule the interaction. The description is done from a global point of view, encompassing the expected behavior of all the participants. Each participant is supposed to use the global definition to build and test solutions that conform to it.

In this work, we focus on the problem of selecting existing services that have to play the roles of a given choreography. This task implies verifying two things: the *conformance* of the service to the specification of a role of interest, and that the use of that service allows the achievement of the *goal*, that caused its search. *Conformance* guarantees the interoperability of the service with the players of the other roles [8], [9], [10] by guaranteeing that the message exchange will produce correct and accepted conversations. The *goal* that caused the search of a service is a condition that should hold after the whole interaction has taken place. It is not tied to the descriptions of some service operation but it is a *global condition* that should hold in the final state, obtained after the conclusion of the conversation/interaction. In a framework in which it is possible to reason on operation preconditions and effects, and where an appropriate specification of the choreography is given, it becomes possible to design services which have a much higher degree of autonomy w.r.t. existing ones and whose behavior resembles more closely the behavior of autonomous agents. In particular, a service can decide whether playing a role by reasoning on the effects of playing the role and see whether it

can achieve a *goal* of interest by doing so. The achievement of the goal depends on the operation sequence because each operation can influence the executability and the outcomes of the subsequent ones. Many of the operations, however, are offered by the partners in the interaction which, at the time when the service reasons about the choreography, they are still unknown. The reasoning process must, therefore, use the specifications of the operations that will be supplied by the partners, specifications which are included in the choreography (that we call unbound operations). A selection process will link unbound operations with operations offered by existing services, and it does so by applying some kind of (possibly flexible) match. In [11], however, we showed that performing a match operation by operation, by applying the definitions in [12], *does not* preserve the global goal. Therefore, the match-making process, that is applied to discover services, should not only focus on local properties of the single operations, e.g. IOPEs, but it should also consider constraints that derive from the global schema of execution, which is given by the choreography. In this paper we extend the results achieved in [11], limited to the so-called plugin match, to the class of re-use ensuring matches [13]. To this aim, inspired by [1], [2], [14], we exploit an action-based representation of the specifications of the operations of a service: each operation is described in terms of its preconditions and effects, as in [15], [16], without taking into account the ontology layer which is not functional to the aims of the work. This representation supplies the mechanisms and the tools for reasoning on compositions of services, as described in choreographies; in particular, it supplies a representation of states and an execution model that can be reasoned about.

The paper is organized as follows. Section II sets the representation of services and of choreographies that we adopt. Section III discusses various kinds of match and reports our proposal for producing conservative matches. A running example is distributed along the pages to better explain the proposed notions and mechanisms. Conclusions end the paper.

II. A THEORETICAL FRAMEWORK FOR REPRESENTING AND REASONING ABOUT SERVICES

In this section, we briefly summarize the notation that we use to represent services, introduced in [16], and we discuss 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. In this perspective, 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 $\mathcal{B}$, to represent the beliefs an entity has about the world: $\mathcal{B}f$ means that the fluent f is known to be true, $\mathcal{B}\neg f$ means that the fluent f is known to be false. A fluent f is undefined when both $\neg\mathcal{B}f$ and $\neg\mathcal{B}\neg f$ hold ($\neg\mathcal{B}f \wedge \neg\mathcal{B}\neg f$). For expressing that a fluent f is undefined, we write $u(f)$. Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*.

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 $\langle \mathcal{S}, \mathcal{P} \rangle$, where $\mathcal{S}$ 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.

A. Basic Operations

The set $\mathcal{S}$ contains the descriptions of a set of service operations. According to the main languages for representing web services, like WSDL and OWL-S, there are four basic kinds of operations [6] (or atomic processes, when using OWL-S terminology [1]): *one-way*, *notify*, *request-response*, and *solicit-response*. The first two involve a single message exchange. In a one-way operation, a client invokes an operation by sending a message to the service, while by a notification the client receives a message from the service. The other two operations involve the exchange of two messages. Request-response operations are initiated by the invoker of the operation, which sends a message to the service and, after that, waits for a response. In solicit-response operations the order of the messages is inverted: first the invoker waits for a message from the service and then it sends an answer.

An operation is described in terms of its *executability preconditions* and *effects*, the former being 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 being a set of fluents (introduced by the keyword **causes**) which will be added to the service state after the operation execution. Formalized in these terms, operations, when executed, trigger a revision process on the actor's beliefs. Since we describe web services from a *subjective* point of view, we distinguish between the case when the service is either the initiator (the operation *invoker*) or the servant of an operation (the operation *supplier*) by further decorating the operation name with a notation inspired by [14]. With reference to a specific service, $operation^{\gg}$ 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 view of operations that is used by *invoker* is given in terms of the operation inputs, outputs, preconditions, and effects as usual for semantic web services [1]. In the next part of this section, inputs and outputs are represented as single messages for simplicity but the representation can easily be extended to sets of exchanged data, as in Example (1). Preconditions P_s and effects E_s are respectively the conditions required by the operation in order to be invoked, and the expected effects that result from the execution of the operation. For what concerns the view of the *supplier*, also in this case the operation is described in terms of its inputs and outputs. Moreover, we also represent a set of conditions that enable the executability

of the operation (R_s, *requirements*) and that constitute the *side effects*, S_s. 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 the four kinds of basic operations. For each operation we report both views.

One-Way, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$
(b) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$
(c) $\mathsf{operation}^{\gg}_{ow}(m_{in})$ **causes** E_s

In one-way operations, the invoker requests an execution which involves sending an information m_{in} to the supplier; the invoker must obviously know the information to send before the invocation (a). The invoker can execute the operation only if the preconditions to the operations 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 that it has sent an information to the supplier (b). Using OWL-S terminology, m_{in} represents the *input* of the operation, while P_s and E_s are its preconditions and effects. One-way operations have no output.

One-Way, supplier point of view:
(a) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **possible if** R_s
(b) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **causes** $\mathbf{B}^{Offerer} m_{in}$
(c) $\mathsf{operation}^{\ll}_{ow}(m_{in})$ **causes** S_s

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 (a). The execution of the operation causes the fact that the supplier will know the information sent by the invoker (b). We also allow the possibility of having some side effects on the supplier's state. These effects are not to be confused with the operation effects described by IOPE, and have been added for the sake of completeness.

Notify, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **possible if** P_s
(b) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$
(c) $\mathsf{operation}^{\gg}_{n}(m_{out})$ **causes** E_s

In notify operations, the invoker requests an execution which involves receiving an information m_{out} from the supplier. The invoker can invoke the execution of the operation only if the preconditions to the operations 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 the received information (b). Using OWL-S terminology, m_{out} represents the *output* of the operation, while P_s and E_s are its preconditions and effects. Notify operations have no input.

Notify, supplier point of view:
(a) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **possible if** $\mathbf{B}^{Offerer} m_{out} \wedge R_s$
(b) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **causes** $\mathbf{B}^{Offerer} sent(m_{out})$
(c) $\mathsf{operation}^{\ll}_{n}(m_{out})$ **causes** S_s

The supplier must know the information to send and must meet the requirements R_s (a). The execution of the operation simply causes the fact that the supplier will know that it has sent some information to the invoker (b). As above, we allow the possibility of having some side effects on the supplier's state (c).

Request-response, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **possible if** $\mathbf{B}^{Invoker} m_{in} \wedge P_s$
(b) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$
(c) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$
(d) $\mathsf{operation}^{\gg}_{rr}(m_{in}, m_{out})$ **causes** E_s

In request-response operations, the invoker requests an execution which involves sending an information m_{in} (the input, according to OWL-S terminology) and then receiving an answer m_{out} from the supplier (the output in OWL-S). The invoker can execute the operation only if the preconditions P_s are satisfied in its current state and if it owns the information to send (a). The execution of the invocation brings about the effects E_s (d), and the fact that the invoker knows that it has sent the input m_{in} to the supplier (b). One further effect of the execution is that the invoker knows the answer returned by the operation (c). This representation abstracts away from the actual message exchange mechanism, which is implemented. Our aim is to reason on the effects of the execution on the mental state of the parties [15].

Request-response, supplier point of view:
(a) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **possible if** R_s
(b) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} m_{in}$
(c) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} m_{out}$
(d) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} sent(m_{out})$
(e) $\mathsf{operation}^{\ll}_{rr}(m_{in}, m_{out})$ **causes** S_s

As for one-way operations, the supplier has the requirements R_s to the operation execution (a). It receives an input m_{in} from the invoker (b). The execution of the operation produces an answer m_{out} (c), which is sent to the invoker (d). As usual, we allow the possibility of having some side effects on the supplier's state. 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.

Solicit-response, invoker point of view:
(a) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **possible if** P_s
(b) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{out}$
(c) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} m_{in}$
(d) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Invoker} sent(m_{in})$
(e) $\mathsf{operation}^{\gg}_{sr}(m_{in}, m_{out})$ **causes** E_s

In solicit-response operations, the invoker requests an execution which involves receiving an information m_{out} (the output,

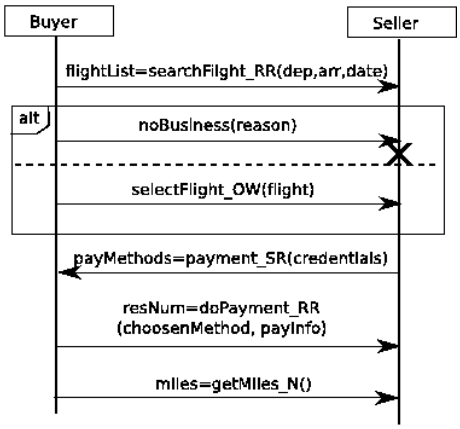

Fig. 1. An example of a simple interaction protocol, for reserving a flight, expressed as a UML sequence diagram.

according to OWL-S terminology) and then sending a message m_{in} to the supplier (the input in OWL-S). The invoker can execute the invocation only if the preconditions P_s are satisfied in its current state (a). The execution of 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).

Solicit-response, supplier point of view:

(a) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **possible if** $\mathbf{B}^{Offerer} m_{out} \wedge R_s$
(b) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} sent(m_{out})$
(c) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** $\mathbf{B}^{Offerer} m_{in}$
(d) $\mathsf{operation}^{\ll}_{sr}(m_{in}, m_{out})$ **causes** S_s

As for notify operations, the supplier must know the information to send and to fulfill the requirements R_s (a). The execution of the operation causes the fact that the supplier will know that it has sent some information to the invoker (b). Moreover, it produces also the knowledge of the information m_{in} received by the invoker (c). As above, we allow the possibility of having some side effects on the supplier's state (d).

Example 1: As an example, let's consider the searchFlight operation of the flight reservation protocol depicted in Figure 1, which is offered by a *seller* and can be invoked by a *buyer* to search information about flights with given departure (dep) and arrival locations (arr) plus the date of departure ($date$). From the point of view of the buyer, the operation, which is of kind request-response, is:

(a) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **possible if** $\mathbf{B}^{buyer} dep \wedge \mathbf{B}^{buyer} arr \wedge \mathbf{B}^{buyer} date \wedge \mathbf{B}^{\mathbf{buyer}} \neg \mathbf{sellingStarted}$
(b) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{buyer} sent(dep) \wedge \mathbf{B}^{buyer} sent(arr) \wedge \mathbf{B}^{buyer} sent(date)$
(d) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{buyer} flightList$
(c) $\mathsf{searchFlight}^{\gg}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{\mathbf{buyer}} \mathbf{sellingStarted}$

The *inputs* of the operation are dep, arr, and $date$, while the *output* is $flightList$. In this case the set P_s contains only the belief $\mathbf{B}^{buyer} \neg sellingStarted$ (in bold text above) while the set E_s of effects contains the belief $\mathbf{B}^{buyer} sellingStarted$ (in bold text as well).

From the point of view of the supplier, instead, the operation is represented as:

(a) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **possible if** $true$
(b) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} dep \wedge \mathbf{B}^{seller} arr \wedge \mathbf{B}^{seller} date$
(c) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} flightList$
(d) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} sent(flightList)$

In this case the sets R_s and S_s of requirements and side effects are empty. The operation expects as input the departure and arrival locations and the date of the flight, and it produces a $flightList$, which it sends to its customer, so after the operation the belief $\mathbf{B}^{seller} sent(flightList)$ will be in its belief state. □

Last but not least, a service can also have internal operations, which can be included in its policy but are not visible from outside. Each operation is represented again as an atomic action, specified by its *preconditions* and its *effects*. Formally, it is defined as:

$\mathsf{operation}(content)$ **causes** E_s
$\mathsf{operation}(content)$ **possible if** P_s

where E_s and P_s, denote respectively the fluents, which are expected as effect of the execution of an operation and the precondition to its execution, while $content$ denotes possible additional data that is required by the operation. Notice that such operations can also be implemented as invocations to other services.

B. Composite operations

$\mathcal{P}$ encodes the complex behavior of the service; it is a collection of clauses of the kind:

$$p_0 \ \textbf{is} \ p_1, \ldots, p_n$$

where p_0 is the name of the procedure and p_i, $i = 1, \ldots, n$, is either an atomic action (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, and their definitions can be non-deterministic as in Prolog.

A *choreography* is made of a set of interacting *roles*, a role being a subjective view of the interaction that is encoded. When a service plays a role in a choreography, its policy will contain some operations which are not of the service itself but belong to some other role of the choreography, with which it interacts. In other words, $\mathcal{S}$ can be partitioned in two sets: a set of bound operations and a set of *unbound operations*, that must be supplied by some counterpart(s). Until the counterpart service(s) is (are) not defined, such operations will be those

specified in the choreography. Such operations will be offered by the interlocutors as $\gg$ operations. We assume that they are represented in a way that is homogeneous with the representation of operations, i.e. by means of *preconditions* and *effects*. The binding will be possible only when the partners in the interaction will be found.

The fact that a service is taking a given role in the choreography is due, in our proposal, to the fact that it knows that a certain goal condition will be true after the execution of the role. When a possible partner is identified for the latter role, after the binding has taken place, it is necessary to check if the goal condition is preserved. The reasons for which this could not happen are explained in the following section; hereafter, we formalize the notion of *substitution* that we interpret as the binding.

Let $S_d = \langle \mathcal{S}, \mathcal{P} \rangle$ be a service description, and let $\mathcal{S}_u$ be a subset of $\mathcal{S}$, containing unbound operations that are to be supplied by a same counterpart S_i. Let $\mathcal{S}_{S_i}$ be the set of operations in S_i that we want S_d to use, binding them to $\mathcal{S}_u$. We represent the binding by the substitution $\theta = [\mathcal{S}_{S_i}/\mathcal{S}_u]$ applied to S_d, i.e.: $S_d\theta = \langle \mathcal{S}\theta, \mathcal{P}\theta \rangle$, where every element of $\mathcal{S}_u$ is substituted by/bound to an element of $\mathcal{S}_{S_i}$. Notice that not all elements of $\mathcal{S}_{S_i}$ are, instead, necessarily bound. An example is reported in *Example* 2.

Example 2: As an instance, here we report the definition of the buyTicket procedure of the flight company example. The procedure will encode a role in a choreography if all of the involved operations are unbound. It will, instead, encode a service behavior when all of its operations are bound to those offered by one or more services that act as interlocutors.

(a) buyTicket **is**
searchFlight$^{\gg}_{rr}((dep, arr, date), flightList)$;
evaluateAndBuy
(b) evaluateAndBuy **is**
noBusiness$^{\gg}_{ow}(reason)$
(c) evaluateAndBuy **is**
selectFlight$^{\gg}_{ow}(flight)$;
payMethods$^{\ll}_{sr}(payMethods, credentials)$;
doPayment$^{\gg}_{rr}((chosenMethod, payInfo), resNum)$;
getMiles$^{\gg}_{n}(miles)$

This procedure encodes the behaviour of the buyer of a flight ticket, be it a role or a specific service. First, it invokes an operation for searchig a flight (searchFlight$^{\gg}_{rr}((dep, arr, date), flightList)$), and it evaluates the result (evaluateAndBuy). The evaluation can give either a negative outcome, hence the interaction is interrupted (noBusiness$^{\gg}_{ow}(reason)$) or the interaction continues with the flight selection (selectFlight$^{\gg}_{ow}(flight)$), the payment (payMethods$^{\ll}_{sr}(payMethods, credentials)$ and doPayment$^{\gg}_{rr}((chosenMethod, payInfo), resNum)$) and, at the end, the client is notified about the obtained miles (getMiles$^{\gg}_{n}(miles)$). □

C. Reasoning on goals

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

$$Fs \ \ \mathbf{after} \ \ p$$

where Fs is the goal (represented as a conjunction of fluents), that we wish to hold after the execution of a policy p. Checking if a formula of this kind holds corresponds 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. This sequence corresponds to an execution trace of the procedure and can be seen as a plan to bring about the *goal* Fs. This form of reasoning is known as *temporal projection*. Temporal projection fits our needs because, as mentioned in the introduction, in order to perform the selection we need a mechanism that verifies if a goal condition holds after the interaction with the service has taken place. Fs is the set of facts that we would like to hold "after" p.

Let $S_d = \langle \mathcal{S}, \mathcal{P} \rangle$ be a service description. The application of temporal projection to $\mathcal{P}$ returns, if any, an execution trace, that makes a goal of interest become true. Let us, then, consider a procedure p belonging to $\mathcal{P}$, and denote by G the query $Fs \ \ \mathbf{after} \ \ p$. Given a state S_0, containing all the fluents that we know as being true in the beginning, we denote the fact that G is successful in S_d by:

$$(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G$$

The execution of the above query returns as a side-effect an *execution trace* σ of p. The execution trace σ is *linear*, i.e. a terminating sequence $a_1, \ldots, a_n$ of atomic actions.

Example 3 (Flight-purchase, second part): Let us suppose that the initial state of the service $b1$ is $S_0 = \{\mathbf{B}^{buyer} dep, \mathbf{B}^{buyer} arr, \mathbf{B}^{buyer} date, \mathbf{B}^{buyer} deferredPaymentPossible, \mathbf{B}^{buyer} \neg sellingStarted\}$, (all the other fluents truth value is "unknown"). This means that $b1$ assumes a date, a departure location, an arrival location, the fact that it is possible to defer the payment to the departure (at a desk at the airport), and that no selling process has started yet. The goal of $b1$ is to achieve the following condition: $G = \{\mathbf{B}^{buyer} sellingComplete, \mathbf{B}^{buyer} resNum\}$ **after** buyTicket Intuitively, the buyer expects that, after the interaction, it will have a reservation number as a result.

By reasoning on its policy and by using the definitions of the unbound operations that are given by the choreography, $b1$ can identify an execution trace, that leads to a state where G holds:

$\sigma =$ searchFlight$^{\gg}_{rr}((dep, arr, date), flightList)$;
selectFlight$^{\gg}_{ow}(flight)$;
payMethods$^{\ll}_{sr}(payMethods, credentials)$;
doPayment$^{\gg}_{rr}((chosenMethod, payInfo), resNum)$;
getMiles$^{\gg}_{n}(miles)$

This is possible because in a declarative representation specifications are executable. Moreover notice that this execution

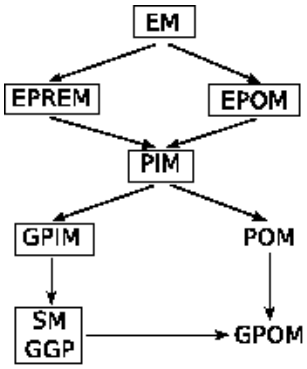

Fig. 2. The lattice of the local matches: on top the strongest; names in a box are re-use ensuring matches; SM and GGP are in same box because logically equivalent.

does not influence the belief about the deferred payment, which persists from the initial through the final state and is not contradicted. □

III. Conservative, re-use ensuring matches

When the matching process is applied for selecting a service that should play a role in a (partially instantiated) choreography, the desire is that the substitution (of the service operations to the specifications contained in the choreography) preserves the properties of interest. In [11] we have formalized this notion in the following way:

Definition 1 (Conservative substitution): Let us consider a service $S_i = \langle \mathcal{S}, \mathcal{P} \rangle$ playing a role R_i in a given choreography, and a query G such that, given an initial state S_0,

$$(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G \text{ w.a. } \sigma$$

Consider a substitution $\theta = [\mathcal{S}_{S_j}/\mathcal{S}^{\sigma}_{u(R_j)}]$, where $\mathcal{S}^{\sigma}_{u(R_j)} = \{o_u \in \mathcal{S} \mid o \text{ occurs in } \sigma\}$ is the set of all unbound operations that refer to another role R_j, $j \neq i$, of the same choreography, that are used in the execution trace σ. θ is conservative when the following holds:

$$(\langle \mathcal{S}\theta, \mathcal{P}\theta \rangle, S_0) \vdash G \text{ w.a. } \sigma\theta$$

□

In the above definition, θ can be any kind of association between the operations of a service with the unbound operations described in a choreography. 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 [12] on software components, and surveyed in [17].

Given a software component S, with precondition S_{pre} and postcondition S_{post}, and a specification (or query, in the match-making community) Q, with precondition Q_{pre} and postcondition Q_{post}, the most important kinds of relaxed match between Q and S are:

- EM (*Exact Pre/Post Match*): $Q_{pre} \Leftrightarrow S_{pre} \wedge Q_{post} \Leftrightarrow S_{post}$
- EPREM (*Exact Pre Match*): $Q_{pre} \Leftrightarrow S_{pre} \wedge S_{post} \Rightarrow Q_{post}$
- EPOM (*Exact Post Match*): $Q_{pre} \Rightarrow S_{pre} \wedge Q_{post} \Leftrightarrow S_{post}$
- PIM (*Plugin Match*): $Q_{pre} \Rightarrow S_{pre} \wedge S_{post} \Rightarrow Q_{post}$
- POM (*Plugin Post Match*): $S_{post} \Rightarrow Q_{post}$
- GPIM (*Guarded Plugin Match*, a.k.a. Weak-Plugin [18]): $Q_{pre} \Rightarrow S_{pre} \wedge ((S_{pre} \wedge S_{post}) \Rightarrow Q_{post})$
- SM (*Satisfies Match*, a.k.a. relaxed plug-in in [13], plug-in compatibility [19]): $Q_{pre} \Rightarrow S_{pre} \wedge (Q_{pre} \wedge S_{post} \Rightarrow Q_{post})$
- GPOM (*Guarded Post Match*, a.k.a. Weak-Post [18]): $((S_{pre} \wedge S_{post}) \Rightarrow Q_{post})$
- GGP (*Guarded-Generalized Predicate*): $(Q_{pre} \Rightarrow S_{pre}) \wedge ((S_{pre} \Rightarrow S_{post}) \Rightarrow (Q_{pre} \Rightarrow Q_{post}))$

The different matches can be organized according to a lattice [17], that we have reported in Fig. 2. On top, there is *Exact pre/post match*, which states the equivalence of Q and S. Moving down in the lattice weaker and weaker match conditions are found. For instance, in *Plugin match* S must only be behaviorally equivalent to Q when plugged-in to replace Q. *Plugin post match* relaxes the former: only the postcondition is considered. *Guarded matches* focus on guaranteeing that the desired postcondition holds when the precondition of S holds, not necessarily in general.

In our application domain, Q is an *unbound operation*, while S is an *operation*. For short, we decorate substitutions with an acronym denoting the applied match (e.g. θ_{EM} is a substitution obtained by applying the exact match, θ_{PIM} by applying the plugin match, etc.). All these matches have been defined for the retrieval of single components, and have a *local* nature, i.e. they compare a requirement to a software specification (in our case, an unbound operation) independently of the *context of usage* (in our work, the choreography role). On the other hand, a choreography defines the *global execution context*, in which unbound operations are immersed.

In [11] we have proved that, in general, flexible matches do not satisfy Definition 1. In other words, it is *not guaranteed* that after the substitution of a set of operations, which were selected by applying one of the local flexible matches, to a set of unbound operations in a role, a goal that could previously be achieved is still achievable. In fact, 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}. This is not a problem when the operation is to be used alone but when it 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. The substitution is necessary in order to make the role executable but this transformation should not affect the possibility of reaching the goal, that was demonstrated for the role specification. In [11] we also showed how to enrich the plugin match so to guarantee that the built substitutions are conservative. This is done by taking into

account the *overall structure*, encoded by the choreography.

In the following we extend this result to a wider class of matches; in particular, we show that all matches which are *re-use ensuring*, according to the definition given by Chen and Cheng in [13], can be enriched in order to guarantee the production of conservative matches. Once again, we do this by exploiting only constraints that can be inferred from the choreography, without modifying the local nature of the considered matches.

Definition 2 (Re-use ensuring match [13]): A specification match M is re-use ensuring iff for any S and Q, $M(S,Q) \wedge \{S_{pre}\}S\{S_{post}\} \Rightarrow \{Q_{pre}\}S\{Q_{post}\}$.

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" [20]. Considering the lattice in Figure 2, re-use ensuring matches are all those in a box, while POM and GPOM are not re-use ensuring.

In order to extend the results in [11] to all re-use ensuring matches, we need to recall a few notions given in that paper. Intuitively, we take into account the *dependencies* between operations, which produce as effects fluents, that are used as preconditions by subsequent operations. The idea is to verify that the "causal chains" which allow the execution of the sequence of operations, are not broken after the substitution. The obvious hypothesis is that we have a choreography and that we know that it allows to achieve the goal of interest, i.e. that there is an execution σ, which allows the achievement of the goal. We will use this trace for defining a set of constraints that, whenever satisfied by a substitution obtained by a re-use 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.

Let us start with the notions of dependencies between operations and dependency sets for fluents. Consider a service description $S = \langle \mathcal{S}, \mathcal{P} \rangle$, and suppose that, given the initial state S_0, the goal $G = Fs$ **after** p succeeds, thus obtaining as answer the successful sequence of operations $\sigma = a_1; a_2; \ldots; a_n$, which is an execution trace of p. We denote by $\overline{\sigma}$ the sequence of operations $a_0; a_1; a_2; \ldots; a_n; a_{n+1}$, where a_0 and a_{n+1} are two *fictitious* operations that will be used respectively to represent the initial state S_0 and the set of fluents Fs, which must hold after σ. That is, we assume a_0 has no precondition and $E_s(a_0) = S_0$, and that a_{n+1} has no effect but $P_s a_{n+1} = Fs$.

Consider two indexes i and j, such that $j < i$, $i, j = 0, \ldots, n+1$. We say that *in* $\overline{\sigma}$ the operation a_i *depends on* a_j *for the fluent* $\mathcal{B}l$, written $a_j \rightsquigarrow_{\langle \mathcal{B}l, \overline{\sigma} \rangle} a_i$, iff $\mathcal{B}l \in E_s(a_j)$, $\mathcal{B}l \in P_s a_i$, and there is not a k, $j < k < i$, such that $\mathcal{B}l \in E_s(a_k)$. Given a fluent $\mathcal{B}l$ and a sequence of operations σ, we can, therefore, define the *dependency set* of $\mathcal{B}l$ as $\mathsf{Deps}(\mathcal{B}l, \sigma) = \{(j,i) \mid a_j \rightsquigarrow_{\langle \mathcal{B}l, \overline{\sigma} \rangle} a_i\}$.

Let $[s/o_u]$ be a specific substitution of a service operation s to an unbound operation o_u, that is contained in θ, we say that a fluent $\mathcal{B}l \in E_s(s) - E_s(o_u)$ (i.e. an additional effect of s w.r.t. the effects of o_u) is an *uninfluential fluent* w.r.t. the sequence $\sigma\theta$ iff for all pairs $(j,i) \in \mathsf{Deps}(\mathcal{B}\neg l, \sigma)$, identifying by k the position of o_u in σ, 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. Note that σ and $\sigma\theta$ have the same length and are identical as sequences of operations but for the fact that in the latter the selected service operations substitute unbound operations. For this reason, we can reduce to reasoning on σ for what concerns the operation positions.

Definition 3: A substitution θ is called *uninfluential* iff for any substitution $[s/o_u]$ in θ, all beliefs in $E_s(s) - E_s(o_u)$ are uninfluential fluents w.r.t. σ.

Proposition 1: Let M be a re-use ensuring match, any substitution θ_M that is uninfluential is also conservative.

Proof: The proof is by absurd and it uses the proof theory introduced in [21]. Let us assume that $(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G$ w.r.a. σ but $(\langle \mathcal{S}\theta_M, \theta_M, \mathcal{P}\theta_M \rangle, S_0) \nvdash G$ w.r.a. $\sigma\theta_M$. Therefore, 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 $\sigma = a_0, a_1, \ldots, a_{i-1}, a_i, \ldots, a_n$ and $F \in P_s(a_i)$, 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 $F \in E_s(a_j)$ 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 $F \notin E_s(a_j\theta_M)$ 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 since 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 $E_s(a_i)$. 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 θ_M is an uninfluential substitution. ■

From the above proposition and the construction of dependency sets, it is easy to show that the following theorem holds.

Theorem 1: Let M be a re-use ensuring match, $S_i = \langle \mathcal{S}, \mathcal{P} \rangle$ be a service which plays a role R_i in a given choreography, and G a query such that, $(\langle \mathcal{S}, \mathcal{P} \rangle, S_0) \vdash G$ w.r.a. σ, where S_O is the initial state. Let θ_M be a substitution for all unbound operations of S_i that refer to another role R_j played by the service S_j, $j \neq i$. The problem of determining whether θ_M is conservative w.r.t. G is decidable.

Example 4: Let us now consider the goal and the service description specified in the previous examples, and suppose that the operation payMethod is defined in this way:

(a) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **possible if**
$\mathbf{B}^{buyer} credentials \wedge$
$\mathbf{B}^{buyer} mustPay(flight)$
$\wedge \mathbf{B^{buyer}}\mathbf{deferredPaymentPossible}$

(b) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **causes**
$\mathbf{B}^{buyer} sent(credentials)$

(c) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **causes**
$\mathbf{B}^{buyer} payMethods$

(d) $\mathsf{payMethods}^{\ll}_{sr}(payMethods, credentials)$ **causes**

$\top$

In particular, the operation has, as a precondition, the possibility of pay the ticket directly at the airport ($\mathbf{B}^{buyer} deferredPaymentPossible$).

Let us now consider a service, that is a candidate to play the role of *Seller*, which is equivalent to the role specification but for the operation that implements searchFlight, which is specified as:

(a) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **possible if** $true$

(b) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} dep \wedge \mathbf{B}^{seller} arr \wedge \mathbf{B}^{seller} date$

(c) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} flightList$

(d) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{B}^{seller} sent(flightList)$

(e) $\mathsf{searchFlight}^{\ll}_{rr}((dep, arr, date), flightList)$ **causes** $\mathbf{\neg B^{buyer} deferredPaymentPossible}$

This operation has an additional effect, w.r.t. to the corresponding unbound operation, that is it negates the possibility of paying the ticket at the airport ($\neg \mathbf{B}^{buyer} deferredPaymentPossible$). Despite this, the service matches with the unbound operation according to many of the re-use ensuring matches (e.g. EPREM, PIM, GPIM, SM). This additional effect is not uninfluential because it prevents the executability of the operation payMethod, as defined above.

If the additional effect were, for instance, that $\mathcal{B}veg\_meals$, supplying an additional information concerning the availability of vegetarian meals, the achievement of the goal would not be compromised and the selection would be allowed. □

IV. Conclusions

In this work we extended the proposal in [11] by proving that for any re-use ensuring match, as defined in [13], it is decidable to verify that the obtained substitutions are conservative w.r.t. a goal that is proved by using for the unbound operations the specifications provided by the choreogrpahy. This result allows the enrichment of the matches with a test that can be applied at the match execution time, locally, i.e. operation by operation. This is done by taking into account the execution context given by the choreography.

The literature related to matchmaking is wide and it is really difficult to be exhaustive. The matches proposed in [12] have inspired most of the semantic matches for web service discovery. Amongst them, Paolucci et al. [4] propose four degrees of match (exact, plugin, subsumes, and fail). Differently than in our proposal, these matches are computed on the ontological relations of the outputs of an advertisement for a service and a query and are orthogonal to our work. This approach tackles DAML-S representations, in which services are described by means of inputs and outputs. This approach is refined in [5], a work that describes a service matchmaking prototype, which uses a DAML-S based ontology and a Description Logic reasoner to compare ontology-based service descriptions, given in terms of input and output parameters. The matchmaking process, like in [4], produces a discrete scale of degrees of match (Exact, PlugIn, Subsume, Intersection, Disjoint).

WSMO (Web Service Modeling Ontology) [2] is an organizational framework for semantic web services. As such, it does not suggest a specific matching rule, which is up to the specific implementations. However, the authors propose in [22] an approach that is based on [12] and on [5], hence it suffers of the same limits that we have mentioned.

Works like [23], [24] propose approaches for goal-driven service composition based on planning. However, the task is accomplished without reference to any choreography. In particular, in [23] the composition and the semantic reasoning phases (carried on on inputs and outputs) are separated and the latter is performed on a local basis only. In [25], [26] web services are composed by composing their interaction protocols in a social framework, by means of a temporal logic.

References

[1] OWL-S Coalition, "http://www.daml.org/services/owl-s/."

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

[3] B. Meyer, "Applying "design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[4] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Proc. of ISWC '02*. Springer, 2002, pp. 333–347.

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

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

[7] WS-CDL, "http://www.w3.org/tr/ws-cdl-10/."

[8] S. K. Rajamani and J. Rehof, "Conformance checking for models of asynchronous message passing software," in *Proc. of 14th International Conference on Computer Aided Verification, CAV 2002*, ser. LNCS, vol. 2404. Springer, 2002, pp. 166–179.

[9] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based analysis of obligations in web service choreography," in *Proc. of IEEE International Conference on Internet&Web Applications and Services 2006*, 2006.

[10] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, "Choreography and orchestration: a synergic approach for system design," in *Proc. of ICSOC 2005*, 2005.

[11] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Service selection by choreography-driven matching," in *Emerging Web Services Technology*, ser. Whitestein Series in Software Agent Technologies and Autonomic Computing, T. Gschwind and C. Pautasso, Eds. Birkhäuser, September 2008, vol. II, ch. 1, pp. 5–22.

[12] A. M. Zaremski and J. M. Wing, "Specification matching of software components," *ACM Transactions on SEM*, vol. 6, no. 4, pp. 333–369, 1997.

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

[14] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Synthesis of Underspecified Composite e-Service bases on Automated Reasoning," in *Proc. of ICSOC04*. ACM, 2004, pp. 105–114.

[15] 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*, vol. 41, no. 2-4, pp. 207–257, 2004. [Online]. Available: http://www.kluweronline.com/issn/1012-2443

[16] 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*, vol. 70, no. 1, pp. 53–73, 2007.

[17] H. Toth, "On theory and practice of assertion based software development," *Journal of Object Technology*, vol. 4, no. 2, pp. 109–129, 2005.
[18] J. Penix and P. Alexander, "Efficient specification-based component retrieval," *Automated Software Engg.*, vol. 6, no. 2, pp. 139–170, 1999.
[19] B. Fischer and G. Snelting, "Reuse by contract," in *ESEC/FSE-Workshop on Foundations of Component-Based Systems*, 1997.
[20] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.
[21] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming Rational Agents in a Modal Action Logic," *AMAI*, vol. 41, no. 2-4, pp. 207–257, 2004. [Online]. Available: http://www.kluweronline.com/issn/1012-2443
[22] U. Keller, R. L. A. Polleres, I. Toma, M. Kifer, and D. Fensel, "D5.1 v0.1 wsmo web service discovery," WSML deliverable, Tech. Rep., 2004.
[23] M. Pistore, L. Spalazzi, and P. Traverso, "A minimalist approach to semantic annotations for web processes compositions." in *ESWC*, 2006, pp. 620–634.
[24] 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.
[25] L. Giordano and A. Martelli, "Web Service Composition in a Temporal Action Logic," in *Proc. of 4th International Workshop on AI for Service Composition (held in conjunction with ECAI 2006)*, Riva del Garda, August 2006.
[26] A. Martelli and L. Giordano, "Reasoning About Web Services in a Temporal Action Logic," in *Reasoning, Action and Interaction in AI Theories and System*, ser. LNAI. Springer, 2006, no. 4155, pp. 229–246.