The need of capability requirements inside choreographies and interaction protocols

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

ABSTRACT

A typical problem of research in the area of Service-Oriented Systems is the composition of a set of services for executing a complex task. In this paper we face an instance of this problem in which a set of parties (be they peers, agents or systems) have to interact according to a given choreography. In particular, we mean to exploit the role definitions contained in the choreography for realizing interaction policies that can be executed by the involved parties. In this case it is necessary that the choreography captures not only the interactive behavior of the system as a whole but that the role definitions contain also a set of *requirements* of capabilities that the parties should exhibit, where by the term "capability" we mean the skill of doing something or of making some condition become true. Such capabilities have the twofold aim of connecting the interactive behavior to be shown by the role-player to its internal state and of making the policy executable.

1. INTRODUCTION

In various application contexts there is a growing need of being able to compose sets of heterogeneous and independent entities with the general aim of executing a task, whose complexity cannot be handled by a single component. In this framework, it is mandatory to find a flexible way for gluing components, a highly complex problem which encompasses various skills (describing the goal to be achieved, describing the solution in terms of involved entities and of their interactions, identifying within the pool of available entities those which can solve subproblems, etc.), pursued by different scientific disciplines. One solution which is being explored both in the web services research area and in the multi-agent systems (MAS) research area is to compose entities based on "dialogue". In the case of web services, ad hoc languages (e.g. WS-BPEL [19]), have been proposed for building executable composite services based on a description of the flow of information, in terms of the messages that are exchanged by the composed services. On the other hand, the problem of aggregating communicating agents into (open) societies is well-known, and a lot of attention has been devoted to the issues of defining interaction policies, verifying the interoperability of agents based on dialogue, and checking the conformance of policies w.r.t. a global communication protocol [11].

As observed recently [25, 5], the MAS and WS research areas show convergences in the approach by which systems of agents, on a side, and composite services, on the other, are designed, implemented and verified. In both cases it is in fact possible to distinguish two levels: a global and abstract view of the system as a whole, which is independent from the specific agents/peers which will take part to the interaction (the design of the system), and the implementation of the system in which the specific entities that will interact are identified. In the case of MASs [14] the design level often corresponds to a shared interaction protocol. In a services oriented scenario this level corresponds to a general *choreography* of the system, in which a set of roles are captured together with their interactions by means of ad hoc representation languages (e.g. WS-CDL). On the other hand, the interactive behavior of a specific party/peer involved in the interaction is to be given in some executable language (e.g. WS-BPEL).

In this proposal, we will consider choreographies as *shared knowledge* among the parties. We will, then, refer to choreographies as to *public* and non-executable specifications. The same assumption cannot be made for what concerns the interactive behavior of specific parties (be they service oriented entities, peers or agents). The actual behavior of a peer will, then, be considered as being private (i.e. non-inspectable from outside). Nevertheless, if we are interested in coordinating the interaction of a set of parties as specified by a given choreography, we need to associate specific parties to roles. A realistic scenario is that in which an entity might just publish the fact that it acts according to the role "seller" of a public choreography. For interacting with that entity it will, then, be necessary to play another role of the specified choreography (e.g. "customer"). For playing a role described in a choreography a peer must own a policy that is conformant to that role. In this scenario the so called *conformance checking* of a policy w.r.t. choreography guarantees that the peer owning the policy can interoperate with peers playing the other roles in the choreography.

Let us focus on the case when a peer does not have any conformant policy for playing a certain role described in a choreography specification, but it would like to take part to an interaction ruled by the choreography. A possible solution is to define a method for generating, in an automatic way, a conformant policy from the role specification. The role specification, in fact, contains all the necessary information about what sending/receving to/from which peer at which moment. As a first approximation, we can, then, think of translating the role as expressed in the specification language in a policy (at least into a policy *skeleton*) given in an executable language. This is, however, not sufficient. In fact, it is necessary to bind the interactive (observable) behavior that is encoded by the role specification with the internal (unobservable) behavior that the peer must anyway have and with its internal state. For instance, the peer must have some means for retrieving or building the information that it sends. This might be done in several ways, e.g. by querying a local data base or by querying another peer. The way in which this operation is performed is not relevant, the important point is to be sure that in principle the peer can execute it. For completing the construction of the policy, it is necessary to have a means for checking whether the peer can actually play the policy, in other words, if it has the *required capabilities*. This can only be done if we have a specification of which capabilities are required in the choreography itself. The capability verification can be accomplished role by role by the specific party willing to take part to the interaction.

This paper aims at introducing the concept of capability in the global/local system/entity specifications, in such a way that capabilities can be accounted for during the processes that are applied for dynamically building and possibly customizing policies. Section 2 defines the setting of the work. Moreover, a first example of protocol (the well-known FIPA Contract Net protocol), that is enriched with capabilities, is reported. Section 3 introduces our notion of *capability test*, making a comparison with systems in which this notion is implicit. The use of reasoning techniques that can be associated with the capability test for performing a customization of the policy being constructed is also discussed. In Section 4 a possible extension of WS-CDL [26] with capability requirements is sketched. Conclusions follow.

2. CHOREOGRAPHIES, INTERACTION PROTOCOLS AND CAPABILITIES

In the introduction we have sketched a scenario in which a system of interacting parties is described by a choreography at an abstract level, in which specific peers do not yet appear. A choreography is a schema, a set of rules according to which interaction should occur. In this context the problem of verifying whether a party's interaction policy respects a given role specification is extremely relevant. This problem is known as *conformance* test [1, 13, 3]. The conformance test can be a means for guaranteeing a priori the interoperability of a set of peers, each playing one of the roles described by a given choreography [3, 5, 10].

In this work we will focus on the case in which a peer is interested in playing a role in an interaction ruled by a choreography, but it does not have a conformant policy. In order for this to happen, it is necessary that the peer *adopts* a new interaction policy. If this scenario were set in an agent-framework, one might think of enriching the set of behaviors of the agent, which failed the conformance test, by asking other agents to supply a correct interaction policy. This solution has been proposed from time to time in the literature; recently it was adopted in Coo-BDI architectures [2]. CooBDI extends the BDI (*Belief, Desire, Intention*) model in such a way that agents are enabled to cooperate through a mechanism, which allows them to exchange plans and which is used whenever it is not possible to find a plan, for pursuing a goal of interest, by just exploiting the local agent's knowledge. The ideas behind the CooBDI theory have been implemented by means of WS technologies, leading to CooWS agents [8]. Another recent work in this line of research is [24]. Here, in the setting of the DALI language, agents can cooperate by exchanging sets of rule that can either define a procedure, or constitute a module for coping with some situation, or be just a segment of a knowledge base. Moreover, agents have reasoning techniques that enable them to evaluate how useful the new information is. Unfortunately, these techniques cannot be directly imported in the context of Service-oriented Computing. The reason is that, while in agent systems it is not a problem to find out *during* the interaction that an agent does not own all the necessary actions, when we compose entities in a service oriented scenario it is fundamental that the analogous knowledge is available *before* the interaction among the peers takes place.

Going back to the situation in which a peer failed the conformance test, one might think of using the protocol definition for supplying the entity with a new policy that is obtained directly from the definition of the role that the peer would like to play. A policy skeleton could be directly synthesized in a semi-automatic way from the protocol description. A similar approach has been adopted, in the past, for synthesizing agent behaviors from UML specifications in [17]. In this perspective, a problem arises: protocols only concern *communication patterns*, i.e. the interactions of a peer with others, abstracting from all references to the internal state of the player and from all actions/instructions that do not concern communication. Nevertheless, in our framework we are interested in a policy that the peer will *execute* and, for permitting the execution, it is necessary to express to some extent also this kind of information. The conclusion is that if we wish to use protocols as a basis for policy skeletons, we need to specify some more information, i.e. actions that allow us the access to the peer's internal state. Throughout this work we will refer to such actions as *capability requirements.*

The term "capability" has recently been used by Padgham et al. [20] (the work is inspired by JACK [9] and it is extended in [21]), in the BDI framework, for identifying the "ability to react rationally towards achieving a particular goal". More specifically, an agent has the capability to achieve a goal if its plan library contains at least one plan for reaching the goal. The authors incorporate this notion in the BDI framework so as to constrain an agent's goals and intentions to be compatible with its capabilities. This notion of capability is orthogonal w.r.t. what proposed in our work. In fact, we propose to associate to a choreography (or protocol) specification, aimed at representing an interaction schema among a set of yet unspecified peers, a set of *requirements* of capabilities. Such requirements specify "actions" that peers, willing to play specific roles in the interaction schema, should exhibit. In order for a peer to play a role, some verification must be performed for deciding if it matches the requirements.

In this perspective, our notion of capability resembles more closely (sometimes unnamed) concepts, that emerge in a more or less explicit way in various frameworks/languages, in which there is a need for defining interfaces. One example is Jade [15], the well-known platform for developing multi-agent systems. In this framework policies are supplied as partial implementations with "holes" that the programmer must fill with code when creating agents. Such holes are represented by methods whose body is not defined. The task of the programmer is to implement the specified methods, whose name and signature is, however, fixed in the partial policy. Another example is powerJava [6, 7], an extension of the Java language that accounts for roles and institutions. Without getting into the depths of the language, a role in powerJava represents an interlocutor in the interaction schema. A role definition contains only the implementation of the interaction schema and leaves to the role-player the task of implementing the internal actions. Such calls to the player's internal actions are named "requirements" and are represented as method prototypes.

Checking whether a peer has the capability corresponding to a requirement is, in a way, a complementary test w.r.t. checking conformance. With a rough approximation, when I check conformance I abstract away from the behavior that does not concern the communication described by the protocol of interest, focussing on the interaction with a set of other peers that are involved, whereas checking capabilities means to check whether it is possible to tie the description of a policy to the execution environment defined by the peer.

2.1 An example: the contract net protocol

For better explaining our ideas, we will consider as a choreography the well-known FIPA ContractNet Protocol [12], pinpointing the capabilities that are required to a peer which would like to play the role of *Participant*. Figure 1 reports a UML version of the protocol (dotted rectangles represent capabilities).

ContractNet is used in electronic commerce and in robotics for allowing entities, which are unable to do some task, to have it done. The protocol captures a pattern of interaction, in which the initiator sends a *call-for-proposal* to a set of participants. Each participant can either accept (and send a proposal) or refuse. The initiator collects all the proposals and selects one of them. Figure 1 describes the interactions between the *Initiator* and one of the *Participants*. In this example we can detect three different capabilities, one for the role of Initiator and two for the Participant. Starting from an instance of the concept Task, the Participant must be able to evaluate it by performing the *evaluateTask* capability, returning an instance of the concept Proposal. Moreover, if its proposal is accepted by the *Initiator*, it must be able to execute the task by using the capability *executeTask*, returning an instance of concept Result. On the other side, the Initiator must have the capability *evaluateProposal*

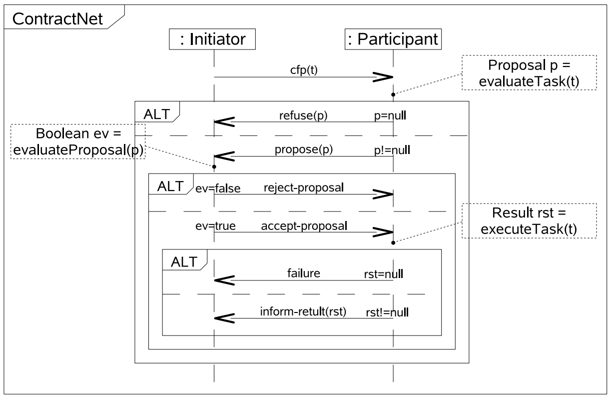

Figure 1: The FIPA ContractNet Protocol, represented by means of UML sequence diagrams, and enriched with capability specifications.

that chooses a proposal among those received from the participants. In order to play the role of Participant a peer will, then, need to have the capabilities *evaluateTask* and *executeTask*, whereas it needs to have the capability *evaluateProposal* if it means to play the role of Initiator. As it emerges from the example, a capability identifies an action (in a broad sense) that might require some inputs and might return a result. This is analogous to defining a method or a function or a web service.

So, it can be meaningful to specify a capability by its name, a description of its inputs and a description of its outputs (see fig 1). However this is not the only possible representation, for instance if we interpret them as actions, it would make sense to represent also their preconditions and effects (or goals).

3. CHECKING CAPABILITIES

In a conformance test we exploit a schema of interaction, the choreography or the protocol, given a priori. The idea that we mean to explore for checking capabilities is to do something analogous for what concerns the internal behavior. In particular, we propose to exploit a description of the required capabilities (see the previous section), which act as connecting points between the external, communicative behavior of the peer and its internal behavior.

The capability test obviously depends on the way in which the policy is developed and therefore it depends on the adopted language. In Jade [15] there is no real capability test because policies already supply empty methods corresponding to the capabilities, the programmer can just redefine them. In powerJava the check is performed by the compiler, which verifies the implementation of a given interface representing the requirements. For further details see [6], in which the same example concerning the ContractNet protocol is described.

In the scenario that we have outlined in the previous section, the capability test is done a priori w.r.t. all the capabilities required by the role specification, however, the way in which the test is implemented is not predefined and can be executed by means of different matching techniques. We could use a simple *signature matching*, like in classical programming languages and in powerJava, as well more flexible forms of matching. We consider particularly promising to adopt *semantic matchmaking* techniques proposed for matching web service descriptions with queries, based on *ontologies* of concepts. In fact semantic matchmaking supports the matching of capabilities with different names, though connected by an ontology, and with different numbers (and descriptions) of input/output parameters. For instance, let us consider the *evaluateProposal* capability associated to the role *Initiator* of the ContractNet protocol (see Figure 1). This capability has an input parameter (a proposal) and is supposed to return a boolean value, stating whether the proposal has been accepted or refused. A first example of flexible, semantics-based matchmaking consists in allowing a peer to play the part of *Initiator* even though it does not have a capability of name *evaluateProposal*. Let us suppose that *evaluateProposal* is a concept in a shared ontology. Then, if the peer has a capability *evaluate*, with same signature of *evaluateProposal*, and *evaluate* is a concept in the shared ontology, that is more general than *evaluateProposal*, we might be eager to consider the capability as matching with the description associated to the role specification.

Semantic matchmaking has been thoroughly studied and formalized also in the Semantic Web community, in particular in the context of the DAML-S [22] and WSMO initiatives [16]. In [22] a form of semantic matchmaking concerning the input and output parameters is proposed. The ontological reasoning is applied to the parameters of a semantic web service, which are compared to a query. The limit of this technique is that it is not possible to perform the search on the basis of a goal to achieve. A different approach is taken in the WSMO initiative [16], where services are described based on their preconditions, assumptions, effects and postconditions. Preconditions concern the structure of the request, assumptions are properties that must hold in the current state, as well as effects will hold in the final state, while postconditions concern the structure of the answer. These four sets of elements are part of the "capability" construct used in WSMO for representing a web service. Moreover, each service has its own choreography and orchestration, although these terms are used in a different way w.r.t. our work. In fact, both refer to subjective views, the former recalls a state chart while the latter is a sequence of if-then rules specifying the interaction with other services. On the other hand, users can express goals as desired postconditions. Various matching techniques are formalized, which enable the search for a service that can satisfy a given goal; all of them presuppose that the goal and the service descriptions are ontology-based and that such ontologies, if different, can be aligned by an ontology mediator. Going back to our focus concerning capability matching, in the WSMO framework it would be possible to represent a "capability requirement", associated with a choreography, as a WSMO goal, to implement the "capabilities" of the specific peer as WSMO capabilities, and then apply the existing matching techniques for deciding whether a requirement is satisfied by at least one of the capabilities of the peer.

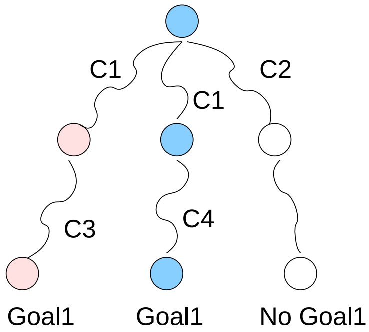

Figure 2: Execution traces for a policy: two traces allow to reach a final state in which *goal1* is true but exploiting different capabilities.

In order to ground our proposal to the reality of Service Oriented Architectures, in Section4, we will discuss an extension of WS-CDL with capability requirements. Our idea is to express such requirements in a general way, in order to enable the use of different matching techniques, which can require different annotations. Thus, we propose an extension of WS-CDL, called WS-CDL+C, where capability requirements can be expressed by input and output parameters as well as by preconditions and goals, by means of semantic annotations. For performing the capability test on this extension, it will be possible to exploit one of the techniques for the semantic matchmaking like the ones cited above.

3.1 Reasoning on capabilities

In the previous sections we discussed the simple case when the capability test is performed w.r.t *all* the capabilities required by the role specification. In this case, based on some description of the required capabilities for a playing the role, we perform the matching among all required and actual service capabilities, thus we can say that the test allows to implement policies that perfectly fit the role, by envisioning all the execution paths foreseen by the role. This is, however, just a starting point. Further customization of the capability test w.r.t. some characteristic or goal of the peer that intend to play a given role can be achieved by combining the test with a reasoning phase on capabilities. For instance, by reasoning on capabilities from the point of view of the party candidate for playing the role, it would be possible to find out policies that implement the role but do not envision all the execution paths and thus do not require the entire list of capabilities associated to the role to be implemented.

Let us take the abstraction of a policy implementing a role w.r.t. all the capabilities required as a procedure with different *execution traces*. Each execution trace corresponds to a branch in the policy. It is likely that only a subset of the capabilities associated to a role will be used along a given branch. As an example, Figure 2 shows three alternative execution traces for a given policy, which contain references to different capabilities: one trace exploits capabilities *C1* and *C3*, the second one exploits *C1* and *C4*, the third one contains only *C2*.

We can think of a simplification of the capability test in which only the execution traces concerning the specific call, that the peer would like to enact, are considered. This set will tell us which capabilities are actually necessary in our execution context (i.e. given the specified input parameter values). In this perspective, it is not compulsory that the party has all the capabilities associated to the role but it will be sufficient that it has those used in this set of execution traces. Consider Figure 2 and suppose that for some given input values, only the first execution trace (starting from left) might become actually executable. This trace relies on capabilities *C1* and *C3* only: it will be sufficient that the peer owns such capabilities for making the *policy call* executable.

Such kind of reasoning could be done by describing the ideal complete policy for an entity aiming at implementing a given role in a declarative language that supports a-priori reasoning on the policy executions. In fact, if a *declarative representation* of the complete policy were given, e.g. see [4], it would be possible to perform a rational inspection of the policy, in which the execution is simulated. By reasoning we could select the execution traces that allow the peer to complete the interaction for the inputs of the given call. Finally we could collect the capabilities used in these traces only (*C1*, *C3*, and *C4* but not *C2*) and restrict the capability test to that subset of capabilities.

Another possible customization task consists on reasoning about those execution traces that, after the execution, make a certain condition become true in the peer internal state. For instance, with reference to Figure 2, two out of the three possible executions lead to a final situation in which *goal1* holds. As a simple example of this case, let us suppose that a peer that wishes to play the role of "customer" with the general goal of purchasing an item of interest from a seller of interest, has a second goal, i.e. to avoid the use of credit cards. This goal can actually be seen as a constraint on the possible interactions. If the policy implementing the complete role allows three alternatives forms of payment (by credit card, by bank transfer and by check), the candidate customer is likely to desire to continue the interaction because some of the alternatives allow reaching the goal of purchasing the item of interest without using credit cards. It can, then, customize the policy by deleting the undesired path. If some of the capabilities are to be used *only* along the discarded execution path, it is not necessary for the candidate customer to have it.

Nevertheless a natural question arises: if I remove some of the possible execution paths of a policy, will it still be conformant to the specification? To answer to this question we can rely on our conformance test. In the specific case of the example, the answer would be positive. It would not be positive if we had a candidate seller that, besides having the general goal of selling items, has the second requirement of not allowing a specific form of payment (e.g. by bank transfer) and deletes the undesired path from the policy. Indeed, a customer that conforms to the shared choreography might require this form of payment, which is foreseen by the specification, but the candidate seller would not be able to handle this case leading to a deadlock.

It is also possible to generalize this approach and selecting the set of the execution traces that can possibly be engaged by a given entity by using the information about the actual capabilities of the services. In fact, having the possibility of inspecting the possible evolutions of an ideal policy implementing the complete role, one could single out those execution traces that require the subset of capabilities that the peer actually can execute. In this way, the policy can be customized w.r.t. the characteristic of the peer, guaranteeing the success under determined circumstances.

Last but not least, the set of capabilities of a peer could be not completely predefined but depending on the context and on privacy or security policies defined by the user. Therefore, I might have a capability which I do not want to use in that circumstance. Also this kind of reasoning can be integrated in the capability test. In this perspective, it would be interesting to explore the use of the notion of *opportunity* proposed by Padmanabhan et al. [21] in connection with the concept of capability (but with the meaning proposed in [20], see Section 1).

4. EXTENDING WS-CDL WITH CAPABILITY REQUIREMENTS

The most important formalism used to represent interaction protocols in the WS domain is WS-CDL (Web Services Choreography Description Language) [26]: an XML-based language that describes peer-to-peer collaborations of heterogeneous entities from a global point of view. In this section, we propose a first step toward the extension to the WS-CDL definition where requirement of capabilities are added in order to enable the automatic synthesis of policies described in the previous sections. We will call this extension WS-CDL+C. Capability requirements are expressed in a general way, in order to enable the of both semantic matchmaking algorithms and reasoning about actions techniques. In particular, we introduce semantic annotations for input and output parameters, that can be exploited by semantic matchmaking like the one described in [22] for performing the capability checking. Moreover we introduce annotations for goals and preconditions of the capability. Goals could be used by a WSMO-like matchmaker, as suggested in section 2. Alternatively, they could be interpreted as effects and, together with preconditions, they could be used by a reasoner for performing the kind of reasoning sketched in Section 3.1. The schema that defines this extension is here omitted for sake of brevity, but can be found at `http://www.di.unito.it/~ alice/WSCDL_Cap_v1.1/`.

In this scenario an operation executed by a peer often corresponds to an invocation of a web service, in a way that is analogous to a *procedure call*. Coherently, we can think of representing the concept of capability in the WS-CDL+C as a new tag element -the tag *capability* (see for instance Figure 3)- which is characterized by its *name*, its *input* and *output* parameters, its *preconditions* and *goals*. Each parameter refers to a variable defined inside the choreography document. The notation `variable="tns:task"` used in Figure 3 is a reference to a variable, according to the definition of WS-CDL. In this manner the variables used in the elements of such description can be used in the whole WS-CDL document in standard ways (like Interaction, Workunit and Assign activities). In particular variables can be used in guard conditions of Workunits inside a Choice activities in order to choose alternative paths (see below for an exam-

```
<silentAction roleType="tns:Participant">
  <capability name="evaluateTask">
    <input>
      <parameter variable="tns:task"/>
    </input>
    <output>
      <parameter variable="tns:proposal"/>
    </output>
    <precondition>
      ?tns:task [name hasValue ?taskName]
      memberOf ont#taskConcept
    </precondition>
    <goal>
      ?tns:proposal [cost hasValue ?cost, time
      hasValue ?time] memberOf
      ont#proposalConcept
   </goal>
   </capability>
 </silentAction>
```

Figure 3: Representing a capability in the extended WS-CDL.

ple). Notice that each variable refers also to a concept in a defined ontology.

A capability represents an operation (a call not a declaration) that must be performed by a role and which is non-observable by the other roles; this kind of activity is described in WS-CDL by *SilentAction* elements. The presence of silent actions is due to the fact that WS-CDL derives from the well-known *pi-calculus* by Milner *et al.* [18], in which silent actions represent the non-observable (or private) behavior of a process . We can, therefore, think of modifying the WS-CDL definition by adding capabilities as child elements of this kind of activity [1]. Returning to Figure 3, as an instance, it defines the capability *evaluateTask* for the role *Participant* of the Contract Net protocol. More precisely, *evaluateTask* is defined within a silent action and its definition comprises its name plus a list of inputs and outputs, plus preconditions and goals. The tags *capability*, *input*, *output*, *precondition* and *goal* are defined in WS-CDL+C. It is relevant to observe that each variable in this description refers to a variable that has been defined in the choreography.

Choreographies not only list the set of capabilities that a peer should have but they also identify the points of the interaction at which such capabilities are to be used. In particular, the values returned by a call to a capability (as a value of an output parameter) can be used for controlling the execution of the interaction. Figure 4 shows, for example, a piece of a choreography code for the role *Participant*, containing a *choice* operator. The *choice* operator allows two alternative executions: one leading to an inform speech act, the other leading to a failure speech act. The selection of which message will actually be sent is done on the basis of the outcome, previously associated to the variable *rst*, of the capability *executeTask*. Only when such variable has a

```
1  <choice>
2    <workunit name="informResultWorkUnit"
3      guard="cdl:getVariable('tns:rst', '', '',
        'tns:Participant') != 'failure' ">
4      <interaction name="informResultInteraction">
5        ...
6      </interaction>
7    </workunit>
8    <interaction name="failureExecuteInteraction">
9      ...
10   </interaction>
11 </choice>
```

Figure 4: Example of how variables in capability requirements can be used in a *choice* operator of a choreography.

non-null value the inform will be sent. The guard condition at line 3 in Figure 4 amounts to determine whether the task that the *Participant* has executed has failed.

To complete the example we sketch in Figure 5 a part of the ContractNet protocol as it is represented in our proposal of extension for WS-CDL. In this example we can detect three different capabilities, one for the role of *Initiator* and two for the role *Participant*. Starting from an instance of the type *Task*, the *Participant* must be able to evaluate it by performing the *evaluateTask* capability (lines 5-10), returning an instance of type *Proposal*. Moreover, it must be able to execute the received task (if its proposal is accepted by the *Initiator*) by using the capability *executeTask* (lines 31-36), returning an instance of type *Result*. On the other side, the *Initiator* must have the capability *evaluateProposal*, for choosing a proposal out of those sent by the participants (lines 18-23).

As discussed before, we can start from a representation of this kind for performing the capability test and checking if a party can play a given role Afterwards it will be possible to synthesize the policy skeleton, possibly customized w.r.t. the capabilities and the goals of the party that is going to play the role. To this aim, a translation algorithm for turning the XML-based specification into an equivalent schema in the execution language of interest is needed.

5. CONCLUSIONS

This work presents a preliminary study aimed at allowing the use of public choreography specifications for automatically synthesizing executable interaction policies for peers that would like to take part to an interaction but that do not own an appropriate policy themselves. To this purpose it is necessary to link the abstract, communicative behavior, expressed at the protocol level, with the internal state of the role player by means of actions that might be non-communicative in nature (capabilities). It is important, in an open framework like the web, to be able to take a decision about the possibility of taking part to a choreography before the interaction begins. This is the reason why we have proposed the introduction of the notion of capability at the level of choreography specification. A capability is the specification of an action in terms of its name, its input, output parameters, preconditions and goals. Given such a

[1]Since in WS-CDL there is not the concept of observable action, capability requirements can describe only silent actions

```
<sequence>
  <interaction name="callForProposalInteraction"> ...
  </interaction>
  <silentAction roleType="Participant">
    <capability name="evaluateTask">
      <input> ...  </input>
      <output> ...  </output>
      <precondition> ... </precondition>
      <goal> ... </goal>
    </capability>
  </silentAction>
  <choice>
    <workunit name="proposeWorkUnit" guard=...  >
      <sequence>
        <interaction name="proposeInteraction">
        </interaction>
        <silentAction roleType="Initiator">
          <capability name="evaluateProposal">
            <input> ...  </input>
            <output> ...  </output>
            <precondition> ... </precondition>
            <goal> ... </goal>
          </capability>
        </silentAction>
        <choice>
          <workunit name="acceptProposalWorkUnit" guard=...>
            <sequence>
              <interaction name="proposeInteraction">
              </interaction>
              <silentAction roleType="Initiator">
                   <capability name="executeTask">
                     <input> ...  </input>
                     <output> ...  </output>
                     <precondition> ... </precondition>
                     <goal> ... </goal>
                   </capability>
              </silentAction>
              <choice>
                <workunit name="informResultWorkUnit"
                  guard=...  >
                  <interaction
                   name="informResultInteraction">
                  </interaction>
                </workunit>
                <interaction
                  name="failureExecuteInteraction">
                </interaction>
              </choice>
            </sequence>
          </workunit>
          <interaction name="rejectProposalInteraction">
          </interaction>
        </choice>
      </sequence>
    </workunit>
    <interaction name="evaluateTaskRefuseInteraction">
    </interaction>
  </choice>
</sequence>
```

Figure 5: A representation of the FIPA ContractNet Protocol in the extended WS-CDL.

description it is possible to apply semantic matching techniques in order to decide whether a peer has the capabilities required for playing a role of interest. In particular, we have discussed the use of semantic matchmaking techniques, such as those developed in the WSMO and DAML-S initiatives [22, 16], for matching web service descriptions to queries.

We have shown how, given a declarative representation of the an ideal policy implementing a role it is possible to apply further reasoning techniques for customizing the policy to the specific characteristic or goals of the entity that will act as a player. Reasoning techniques for accomplishing this customization task are under investigation. In particular, the techniques that we have already used in previous work concerning the personalization of the interaction with a web service [4] seem promising. In that work, in fact, we exploited a kind of reasoning known as *procedural planning*, relying on a logic framework. Procedural planning explores the space of the possible execution traces of a procedure, extracting those paths at whose end a goal condition of interest holds. It is noticeable that in presence of a sensing action, i.e. an action that queries for external input, all of the possible answers are to be kept (they must all lead to the goal) and none can be cut off. In other words, it is possible to cut only paths that correspond to some action that are under the responsibility of the agent playing the policy. The waiting for an incoming message is exactly a query for an external input, as such the case of the candidate seller that does not allow a legal form of payment cannot occur.

Our work is close in spirit to [23], where the idea of keeping separate procedural and ontological descriptions of services and to link them through semantic annotations is introduced. In fact WS-CDL+C can be seen as procedural description of the interaction enriched with capabilities requirements, while semantic annotations of capability requirements enable the use of ontological reasoning for the capability test phase. Presently, we are working at more thorough formalization of the proposal that will be followed by the implementation of a system that turns a role in the proposed extension of WS-CDL into a executable composite service, e.g. a BPEL service. BPEL is just a possibility, any programming language by means of which it is possible to develop web services could be used.

6. ACKNOWLEDGMENTS

This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net), and by MIUR PRIN 2005 “Specification and verification of agent interaction protocols” national project.

7. REFERENCES

[1] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. In *Proc. of the Workshop on Logic and Communication in Multi-Agent Systems, LCMAS 2003*, volume 85(2) of *ENTCS*. Elsevier, 2003.

[2] D. Ancona and V. Mascardi. Coo-BDI: Extending the BDI Model with Cooperativity. In J. A. Leite,

A. Omicini, L. Sterling, and P. Torroni, editors, *Proc. of the 1st Declarative Agent Languages and Technologies Workshop (DALT'03), Revised Selected and Invited Papers*, pages 109–134. Springer-Verlag, 2004. LNAI 2990.

[3] M. Baldoni, C. Baroglio, A. Martelli, and Patti. Verification of protocol conformance and agent interoperability. In F. Toni and P. Torroni, editors, *Post-Proc. of 6th Int. Workshop on Computational Logic in Multi-Agent Systems, CLIMA VI*, volume 3900 of *LNCS State-of-the-Art Survey*, pages 265–283. Springer, 2006.

[4] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *J. of Logic and Algebraic Programming, special issue on Web Services and Formal Methods*, 2006. To appear.

[5] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Verifying the conformance of web services to global interaction protocols: a first step. In M. Bravetti and G. Zavattaro, editors, *Proc. of 2nd Int. Workshop on Web Services and Formal Methods, WS-FM 2005*, volume 3670 of *LNCS*, pages 257–271. Springer, Versailles, France, September, 2005.

[6] M. Baldoni, G. Boella, and L. van der Torre. Bridging Agent Theory and Object Orientation: Importing Social Roles in Object Oriented Languages. In R. H. Bordini, M. Dastani, J. Dix, and A. Seghrouchni, editors, *Post-Proc. of the International Workshop on Programming Multi-Agent Systems, ProMAS 2005*, volume 3862 of *Lecture Notes in Computer Science (LNCS)*, pages 57–75. Springer, 2006.

[7] M. Baldoni, G. Boella, and L. van der Torre. powerjava: Ontologically Founded Roles in Object Oriented Programming Languages. In D. Ancona and M. Viroli, editors, *Proc. of 21st ACM Symposium on Applied Computing, SAC 2006, Special Track on Object-Oriented Programming Languages and Systems (OOPS 2006)*, Dijon, France, April 2006. ACM.

[8] 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.

[9] P. Busetta, N. Howden, R. Ronquist, and A. Hodgson. Structuring bdi agents in functional clusters. In *Proc. of the 6th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL99)*, 1999.

[10] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration: a synergic approach for system design. In *Proc. of 4th International Conference on Service Oriented Computing (ICSOC 2005)*, 2005.

[11] F. Dignum, editor. *Advances in agent communication languages*, volume 2922 of *LNAI*. Springer-Verlag, 2004.

[12] F. for Intelligent Physical Agents. http://www.fipa.org.

[13] F. Guerin and J. Pitt. Verification and Compliance Testing. In H. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 98–112. Springer, 2003.

[14] M. P. Huget and J. Koning. Interaction Protocol Engineering. In H. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 179–193. Springer, 2003.

[15] Jade. http://jade.cselt.it/.

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

[17] M. Martelli and V. Mascardi. From UML diagrams to Jess rules: Integrating OO and rule-based languages to specify, implement and execute agents. In F. Buccafurri, editor, *Proceedings of the 8th APPIA-GULP-PRODE Joint Conference on Declarative Programming (AGP'03)*, pages 275–286, 2003.

[18] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus.* Cambridge University Press, 1999.

[19] OASIS. Business process execution language for web services.

[20] L. Padgham and P. Lambrix. Agent capabilities: Extending BDI theory. In *AAAI/IAAI*, pages 68–73, 2000.

[21] V. Padmanabhan, G. Governatori, and A. Sattar. Actions made explicit in bdi. In *Advances in Artificial Intelligence*, number 2256 in LNCS, pages 390–401. Springer, 2001.

[22] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First International Semantic Web Conference*, 2002.

[23] M. Pistore, L. Spalazzi, and P. Traverso. A minimalist approach to semantic annotations for web processes compositions. In *ESWC*, pages 620–634, 2006.

[24] A. T. S. Costantini. Learning by knowledge exchange in logical agents. In F. Corradini, F. De Paoli, E. Merelli, and A. Omicini, editors, *Proc. of WOA 2005: Dagli oggetti agli agenti, simulazione e analisi formale di sistemi complessi*, Camerino, Italy, november 2005. Pitagora Editrice Bologna.

[25] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed. Life after BPEL? In *Proc. of WS-FM'05*, volume 3670 of *LNCS*, pages 35–50. Springer, 2005. Invited speaker.

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