Web Service Composition - Current Solutions and Open Problems

Biplav Srivastava
IBM India Research Laboratory
Block 1, IIT, New Delhi 110016, India
sbiplav@in.ibm.com

Jana Koehler
IBM Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
koe@zurich.ibm.com

Abstract

Composition of Web services has received much interest to support business-to-business or enterprise application integration. On the one side, the business world has developed a number of XML-based standards to formalize the specification of Web services, their flow composition and execution. This approach is primarily syntactical: Web service interfaces are like remote procedure call and the interaction protocols are manually written. On the other side, the Semantic Web community focuses on reasoning about web resources by explicitly declaring their preconditions and effects with terms precisely defined in ontologies. For the composition of Web services, they draw on the goal-oriented inferencing from planning. So far, both approaches have been developed rather independently from each other. We compare these approaches and discuss their solutions to the problems of modeling, composing, executing, and verifying Web services. We discuss what makes the Web service composition so special and derive challenges for the AI planning community.

Introduction

The growing trend in software architecture is to build platform-independent software components, called *Web services*, that are available in the distributed environment of the Internet. Applications are to be assembled from a set of appropriate Web services and no longer be written manually. Seamless composition of Web services has enormous potential in streamlining business-to-business or enterprise application integration.

The functionality of a Web service needs to be described with additional pieces of information, either by a *semantic annotation* of what it does and/or by a *functional annotation* of how it behaves. The industry views Web services as abstract, standardized interfaces to business processes. The specification of a Web service is expressed in WSDL (Christensen *et al.* 2001), which specifies only the syntax of messages that enter or leave a computer program. In which order messages have to be exchanged between services must be described separately in a flow specification. There are many Web services flow specification languages like BPEL4WS (Curbera & others 2002) and WSCI (Arkin & others 2002). The composition of the flow (i.e., plan) is still manually obtained. Semantic annotations have been widely discussed in the Semantic Web community (Berners-Lee, Hendler, & Lassila 2001) where preconditions and effects of services are explicitly declared in the Resource Description Format (RDF) (RDF 1999) using terms from pre-agreed ontologies.[1] Consequently, understanding the meaning of the messages poses no problem at all in this approach. For composing Web services, the semantic-web community draws on AI planning, which for over three decades, has investigated the problem of how to synthesize complex behaviors given an initial state, an explicit goal representation, and a set of possible state transitions. It is often assumed that a business process or application is associated with some explicit business goal definition that can guide a planning-based composition tool to select the right service (McIlraith & Son 2002).

Unfortunately, we found that explicit goals are usually not available from an industrial perspective. A business process model describes the processing of persistent data objects in discrete process steps. The real "goal" of a business often remains implicit in these models and is rather expressed at a higher level using often using balanced score cards, while the implicit goal of a business process is the correct handling or the creation of data objects manifested in persistent documents. For example, the explicit goal of a travel reservation process is to perfectly organize the travel, while its (more or less) implicit goal is the creation of the required travel documents in some data base.

We believe that understanding this and other related issues will pave the way for research directions in planning to effectively address the Web services composition problem. To that end, we take a realistic application domain for Web services (specifically, trip planning) and highlight to what extent business needs are addressed by the WSDL Web services and the Semantic Web approaches. We investigate how these approaches differ with respect to the modeling, verification, and deployment of services and the respective inference methods and runtime support that they assume. We con-

[1] We will qualify Web services with the prefix *semantic* for Web services expressed in RDF and reserve the term *Web services* for business process interfaces expressed in WSDL.

clude with a discussion of related work and describe an industry-relevant, yet unsolved, practical planning problem.

An Example Scenario

We describe in detail the Web service application scenario of booking travel packages in a travel agency[2] and consider how a simple, closed-world travel example evolves into a dynamic, integrated solution.

In the *closed-world* case, a customer talks to the travel agent who notes the customer's requests and generates a *trip request* document that may contain several needed *flight* and *hotel reservations*. The travel agent performs all bookings and when he is done, he puts the *trip request* either into the *cancelled requests* or the *completed requests* data base, see Figure 1. A completed document is sent to the customer as an answer to his request. If the booking fails, the customer is contacted again and the whole process re-iterates.

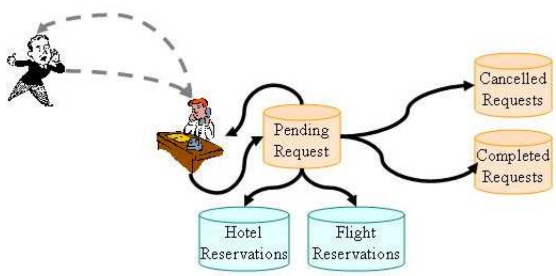

Figure 1: The closed-world travel agency.

Let us assume that our travel agency wants to cooperate with external specialized service providers that offer hotel and flight reservations. The process is now no longer a closed-world solution, but requires to reorganize the entire processing of customer requests. New services have to be integrated and all services must correctly interact with each other.

In this new situation, upon receiving the customer order, the travel agent will still create a *trip request* and derive the required hotel and flight reservations for it. These are now sent to specialized services, which work independently of each other and try to book the desired reservations. The coordination is achieved through the travel agent process to whom the services report their success or failure in making the requested bookings. When both services have been successfully completed, the *confirmation* process step puts all documents together and informs the customer about the completed trip plan. If one of the services fails, the *cancellation* process step tells the other service to put all reservations on hold and contacts the customer to revise his trip.

In the open-world variant of this example scenario, we see properties that are quite typical for real-world planning scenarios:

- a number of processes or agents concurrently and cooperatively try to achieve some goal,
- processes can spawn off other processes at runtime e.g., , for each hotel request a hotel service could be invoked. This number is not known at design time, but only becomes known at runtime.
- the processes partially interleave with each other, synchronize or run fully independent of each other,
- each process exhibits a complex behavior,
- certain planned operations may fail and require to recover from a failed execution.

In the following, we are going to discuss how Web services are used to implement open-world business solutions.

Web Services

Web services are defined as self-contained, modular units of application logic which provide business functionality to other applications via an Internet connection. Web services support the interaction of business partners and their processes by providing a stateless model of "atomic" synchronous or asynchronous message exchanges. These "atomic" message exchanges can be composed into longer business interactions by providing message exchange protocols that show the mutually visible message exchange behavior of each of the partners involved. The issue of how web services are to be described can be resolved in various ways.

Web Services in WSDL

The Web Services Definition language (WSDL) (Christensen *et al.* 2001) is an XML-based language, which specifies a *Web service* by defining messages that provide an abstract definition of the data being transmitted and operations that a Web service provides to transmit the messages. Four types of communication are defined involving a service's operation (endpoint): the endpoint receives a message (one-way), sends a message (notification), the endpoint receives a message and sends a correlated message (request-response), and it sends a message and receives a correlated message (solicit-response). Operations are grouped into port types, which describe abstract end points of a Web service such as a logical address under which an operation can be invoked. A WSDL *message* element defines the data elements of an operation. XML Schema syntax is used to define platform-independent data types which messages can use. Each *message* can consist of one or more *parts*. The *parts* can be compared to the parameters of a function call in a traditional programming language. Concrete protocol bindings and physical address port specifications complete a Web service specification.

We show a sample WSDL fragment for a possible travel agency service. Port type names are simple default strings and operation names illustrate the interacting process steps. For example, "Customer to Create

[2] See http://www.w3.org/2002/04/17-ws-usecase.

Itinerary" is abbreviated with CToCI, "Create Itinerary to Flight Service" is abbreviated with CIToFS, RIToFS stands for "Replan Itinerary to Flight Service".

```
<definitions targetNamespace="http://..."
   xmlns="http://schemas.xmlsoap.org/wsdl/">
 <message name = "OrderEvent"></message>
 <message name = "TripRquest"></message>
 <message name = "FlightRequest"></message>
 <message name = "HotelRequest"></message>
 <message name = "BookingFailure"></message>

 <portType name ="pt1">
   <operation name ="CToCI">
      <input message ="TripRequest"/>
   </operation>
 </portType>
 <portType name ="pt2">
   <operation name ="CIToHS">
      <output message ="HotelRequest"/>
   </operation>
 </portType>
 <portType name ="pt3">
   <operation name ="CIToFS">
      <output message ="FlightRequest"/>
   </operation>
 </portType>
...
 <portType name ="pt9">
   <operation name ="RIToFS">
      <output message ="BookingFailure/>
   </operation>
 </portType>
</definitions>
```

From the definition, once can see that a Web service is viewed like a remote procedure call (RPC). The exact control and data flow that determines when an operation can execute, is provided in a flow composition language like BPEL4WS—we show the flow for the travel example in Section . A service is invoked and it either gets a synchronous or an asynchronous response. In this fragment, all operations are asynchronous, i.e., none of them specifies an input and an output message. The messages are simple syntactic descriptions (usually given as an associated XML schema that we do not show) without any semantics.

Web Services in the Semantic Web

The Semantic Web (Berners-Lee, Hendler, & Lassila 2001) views the World Wide Web as a globally linked database where web pages are marked with semantic annotations. At the core, semantic annotations are assertions about web resources and their properties (example, "A is subclass of B") expressed in the Resource Description Format (RDF) (RDF 1999). An RDF description is a set of triples where each triple is akin to the subject, verb and object of a sentence. Each element of the triple is represented by a Universal Resource Identifier (URI). RDF can be written in multiple notations - XML, Notation3, etc. Along with RDF, one can use RDF Schema (RDFS) to express classes, properties, ranges and documentation for resources and the DAML-S (Ankolenkar & others 2002) ontology to represent further relationships and/or properties like equivalences, lists, and data types. DAML-S has defined *Service* class to model Web services with the properties *presents*, *describedBy* and *supports*. The properties in turn have classes *ServiceProfile*, *ServiceModel* and *ServiceGrounding* as their respective ranges.

For the travel reservation example, we will model the different participating Web services in terms of the DAML-S ontology. Let us assume that our travel agency will use the Web service interface similar to the fictious airline, Bravo Air, whose DAML-S description is publicly available.[3]

The *ServiceProfile* gives a high-level description of the service that can be used to advertise its features and used by clients to select and locate the service from registries. The most important information it contains are the inputs, outputs, preconditions and postconditions of the service. For example, the flight itinerary output from the airline is shown below. It is a specialization of the general flight itinerary concept defined elsewhere[4] and also refers to the round trip resource.

```
<profile:output>
  <profile:ParameterDescription
           rdf:ID="FlightItinerary">
    <profile:parameterName>
        FlightItinerary
    </profile:parameterName>
    <profile:restrictedTo
       rdf:resource=".../concepts.daml
           #FlightItinerary"/>
    <profile:refersTo
       rdf:resource="#roundTrip_In"/>
  </profile:ParameterDescription>
</profile:output>
```

The *ServiceModel* is a detailed description of the service in which it is modeled as a *process*. This description is further sub-divided into a *process model*, which describes the sub-components of the service and a *process control model*, which provides a runtime framework to monitor the execution of the service. In the *process model* description of a composite process, the sub-processes dependencies and interactions can be expressed by *Sequence*, *Split*, *Unordered*, etc. We will show the representation of a static (sub-)flow for the flight Service at Bravo Air in Section .

Finally, the *ServiceGrounding* provides the binding level information of how a client can access the service, e.g., by using SOAP or Java RMI. In the fragment shown below, DAML-S parameters are mapped to WSDL descriptions which in turn have protocol-level binding information.

```
<grounding:wsdlOutputMessageParts
   rdf:parseType="daml:collection">
 <grounding:WsdlMessageMap>
  <grounding:damlsParameter
```

[3]See http://www.daml.org/services/daml-s/0.7/.

[4]See http://www.daml.ri.cmu.edu/ont/DAML-S/ concepts.daml.

```
    rdf:resource=".../FlightItinerary"/>
   <grounding:wsdlMessagePart>
    <xsd:uriReference
       rdf:value=".../availFlightItinerary"/>
   </grounding:wsdlMessagePart>
 </grounding:WsdlMessageMap>
</grounding:wsdlOutputMessageParts>
```

With the Semantic Web infrastructure in place, practical and powerful applications can be written that use annotations and suitable inference engines to automatically discover, execute, compose, and interoperate Web services. While type and consistency checking are currently possible and inferencing with Horn-clauses will soon be available, larger subsets of first-order predicate logic are not yet supported by the Semantic Web, i.e., the degree of logical expressivity as it has been achieved in today's planning languages has not yet been reached.

Discussion

WSDL provides a function-centric description of Web services covering inputs, outputs, and exception handling. The Semantic Web provides a process level description of the service which, in addition to functional information, models the preconditions and postconditions of the process so that the evolution of the domain can be logically inferred. It relies on ontologies to formalize domain concepts which are shared among services.

Modeling Flow Composition

We now discuss how the two approaches address the Web service composition problem. Given the different information that is available to specify a Web service in either approach, it is not very surprising that the solutions to the flow composition problem come out in apparently different (albeit not so divergent) ways, too.

Industry Solution: WSDL + BPEL4WS

We saw the interface description of the travel agent Web service in WSDL. The interactions and message exchanges between the travel agent and its partners—the customer, the flight and hotel services are described in a business protocol specification language (we will concentrate on the standard BPEL4WS proposed by IBM and Microsoft), which specifies the roles of each of the partners and the logical flow of the message exchanges from the perspective of the travel agent process. To make the example more interesting, we assume that the interaction with the customer will also proceed via a Web service interface.

```
<process name ="TripHandling">
  <partners>
    <partner name ="Customer"
           myRole ="TripHandlingAgent"
  serviceLinkType ="ExternalServiceLink"
      partnerRole ="CustomerAgent"/>

    <partner name ="FlightService"
           myRole ="TripHandlingAgent"
  serviceLinkType ="InternalServiceLink"
      partnerRole ="FlightServiceAgent"/>

    <partner name ="HotelService"
           myRole ="tripHandlingAgent"
  serviceLinkType ="InternalServiceLink"
      partnerRole ="HotelServiceAgent"/>
  </partners>

  <containers>  ...  <containers>
  ...
</process>
```

For each WSDL message, a corresponding container to hold that message must be specified.

```
<containers>
  <container name ="OrderEvent"
      messageType ="OrderEventType"/>
  <container name ="TripRequest"
      messageType ="TripRequestType"/>
  <container name ="FlightRequest"
      messageType ="FlightRequestType"/>
  <container name ="HotelRequest"
      messageType ="HotelRequestType"/>
  <container name ="BookingFailure"
      messageType ="BookingFailureType"/>
</containers>
```

The most difficult task for an IT specialist is to specify the logic of the message flow.[5] For this purpose, BPEL4WS provides programming-language like constructs (sequence, switch, while, pick) as well as graph-based links that represent additional ordering constraints on the constructs. The language is fairly complex as the example below illustrates.

The process starts when it receives a trip request from the customer. After the request has been received, hotel and flight request messages can be sent in any order to the two partner services.

```
<sequence>
  <receive partner="Customer"
         portType ="pt1"
        operation ="CToCI"
        container ="OrderEvent">
  </receive>

  <flow>
    <invoke partner ="HotelService"
           portType ="pt2"
          operation ="CIToHS"
     inputContainer ="HotelRequest">
    </invoke>
    <invoke partner ="FlightService"
           portType ="pt3"
          operation ="CIToFS"
     inputContainer ="FlightRequest">
    </invoke>
  </flow>
```

[5]Development effort to support or partially automate this task is under way in the industry, but this goes beyond the scope of our paper.

After the partner services have been invoked, the process waits for the services to send the results of their booking operations, which again can arrive in any order.

```
 <flow>
    <receive partner ="HotelService"
      portType ="pt4"
      operation ="HSToEVAL1"
      container ="HotelRequest">
   </receive>

  <receive partner ="FlightService"
          portType ="pt5"
         operation ="FSToEVAL1"
         container ="FlightRequest">
  </receive>
</flow>
```

After the answers have been received, the process needs to branch depending on whether the trip could be booked or not by the services. This introduces a first `<switch>` construct in the process. We only show a very abstract representation of the condition, which in reality will be a complex XPATH expression on the contents of the arriving message. In the first branch (*ConIToC*), the process needs to inform the customer about the successful completion of his reservation and sends the completed trip request documents. In the second branch (*RIToC*), it needs to inform the customer about the booking failure and decide, which of the services has to be informed about the failure of the other partner. The partners can be informed in any order, which again introduces a `<flow>` construct into the process. Within the flow, another `<switch>` construct is nested that decides which partner is informed.

```
<switch>
  <case condition ="condition1">
    <invoke partner ="Customer"
           portType ="pt6"
          operation ="ConIToC"
     inputContainer ="TripRequest">
    </invoke>
  </case>

  <otherwise>
    <flow>
      <invoke partner ="Customer"
             portType ="pt7"
            operation ="RIToC"
       inputContainer ="BookingFailure">
      </invoke>
      <switch>
        <case condition="condition2">
          <invoke partner ="HotelService"
                 portType ="pt8"
                operation ="EVAL2ToHS"
           inputContainer ="BookingFailure">
          </invoke>
        </case>
        <otherwise>
          <invoke partner ="FlightService"
                 portType ="pt9"
                operation ="EVAL2ToFS"
           inputContainer ="BookingFailure">
          </invoke>
        </otherwise>
      ...
</switch>
```

Each of the partner processes needs to be specified in a similar way and it must be made sure that these specifications fit to each other. We will get back to this problem in Section .

Semantic Web Solution: RDF/DAML-S + Golog/Planning

Let us now try to solve the service composition problem of the travel domain from a Semantic Web perspective. We again focus on one of the partners and consider the static (sub-)flow for the flight reservation at Bravo Air, which is specified in the *process model* of the *ServiceModel*. The service model states that the flight reservation at Bravo Air is a composite process (service) realized by invoking the *sequence* of sub-processes: GetDesiredFlightDetails, SelectAvailableFlight, and BookFlight. Basically, the *ServiceModel* construct of each service will be used to describe its (static) process level description, and the complete specification is comparable to a BPEL4WS process specification.

```
<daml:Class rdf:ID="BravoAir_Process">
 <daml:subClassOf rdf:resource=
     ".../Process.daml#CompositeProcess"/>
  <daml:subClassOf>
   <daml:Restriction>
    <daml:onProperty rdf:resource=
         ".../Process.daml#composedOf"/>
     <daml:toClass>
      <daml:Class>
       <daml:intersectionOf rdf:parseType=
          "daml:collection">
        <daml:Class rdf:about=
          "process:Sequence"/>
         <daml:Restriction>
          <daml:onProperty rdf:resource=
              ".../Process.daml#components"/>
           <daml:toClass>
            <daml:Class>

<process:listOfInstancesOf rdf:parseType=
         "daml:collection">
  <daml:Class rdf:about="#GetDesiredFlightDetails"/>
  <daml:Class rdf:about="#SelectAvailableFlight"/>
  <daml:Class rdf:about="#BookFlight"/>
</process:listOfInstancesOf>

              </daml:Class>
             </daml:toClass>
            </daml:Restriction>
           </daml:intersectionOf>
         </daml:Class>
       </daml:toClass>
     </daml:Restriction>
   </daml:subClassOf>
</daml:Class>
```

In (McIlraith & Son 2002), a method is presented to compose Web services by applying logical inferencing techniques on pre-defined plan templates. The service capabilities are annotated in DAML-S/RDF and then manually translated into Prolog. Now, given a goal description, the logic programming language of Golog (Levesque *et al.* 1997) (which is implemented over Prolog) is used to instantiate the appropriate plan for composing the Web services. Golog is based on the situation calculus and it supports specification and execution of complex actions in dynamical systems. The authors extend it to support sensing actions that can find values of variables at runtime.

```
proc(travel(D1, D2, 0, D)
⊏
   ⊏ bookRAirticket(O, D, D1, D2),
     bookCar(D, D, D1, D2)
   ⊐|
   bookCar(O, O, D1, D2),
   bookHotel(D, D1, D2),
   sendEmail,
   updateExpenseClaim
⊐).
```

Table 1: Travel reservation procedure using Golog. O, D, D1 and D2 stand for Origin, Destination, Departure time and Return time.

Table 1 shows the generalized plan taken from (McIlraith & Son 2002) that is input to their Golog reasoner. The plan specifies that in order to make a travel booking, i.e., to achieve the goal `travel(D1, D2, 0, D)`, either a return air and subsequent car reservation have to be made between the origin and destination, or a direct car reservation has to be made. This will be followed by making hotel reservations, sending an email about the overall reservation to the user and finally updating the expense claim forms. Similar plan/action templates for individual processes like `bookRAirTicket` have to be defined as well.

The Golog reasoner, given the plan and action templates, evaluates non-deterministic choices and executes the plan. Since execution is in the Prolog environment, the non-deterministic choice is actually made according to the default evaluation order (i.e., the order of appearance of literals). Essentially, Golog programs are user-provided plan templates which are customized (bound at runtime) to goal instances. The system uses hand-built wrappers to transform semantic annotations into Golog representations, and vice-versa.

In the following section, we compare the specific characteristics of the two approaches with each other and derive challenges for AI planning research.

Discussion

The specification of the composite service, whether specified in BPEL4WS or DAMLS-S, encodes process information that can be bound to different protocols. One thing to note is that there is significantly more emphasis in BPEL4WS on error handling and message correlation. In BPEL4WS, it is possible to express choice among multiple process activities using the `pick` and `switch` constructs, but the set of choices is predetermined and each activity is conditioned on the occurrence of an event or a specific message contents. A similar construct in DAML-S is the `Choice` construct for selecting a subset of sub-processes from a composite process. The Semantic Web composition solution with RDF/DAML-S and Golog seems to be comparable to the BPEL4WS specification—both allow the customization of the plan execution at runtime, i.e., to select a particular branch of execution or to loop until an exit condition is satisfied, and support the binding of variables to concrete values discovered at runtime.

The industry approach looks at composite services mainly from the runtime perspective of functions, data and control flow. Under this angle, the essential information for reasoning about a service are inputs, outputs and exception handlers. Schemas define and restrict the format of data and define their relationships. Flexibility in dynamically adapting the plan is limited to the specification of binding details at runtime and to executing specific branches in it.

The planning approaches in the Semantic Web are focused on the process-centric description of services as actions that are applicable in states. State transitions are defined based on preconditions of actions and a transitions leads to new states where the effects of the action are valid. They need a representation of state, actions, goals, events and optionally, an ontology of standard terms. The plan can be adapted both offline and online. There is more flexibility in terms of considering different choices of services (plans) based on goals, but the goals are explicitly given.

It is our impression that none of the approaches has developed a true planning solution to the service composition problem so far. Both, BPEL4WS specifications and Golog programs, are written manually and no assembling of complex flows from atomic message exchanges based on a search process takes place. We also see the following characteristics that make current AI planning technology not directly applicable to the service composition problem:

- The action representations can be kept rather simple and only need to model the receiving or sending of a particular message type. However, plans need to contain complex control structures involving loops, nondeterminism, and choice. So far, only the planning as model checking approach has been able to provide initial solutions to the generation of such complex plans (Giunchiglia & Traverso 1999).
- The planning problem cannot really be expected to take place at the level of primitive actions and control structures, but seems to require to take complex plans as building blocks and synthesize multi-partner

interactions from them. We discuss this further below.

- The "objects" manipulated by the actions are typed messages, which have a very rich structure. A typed message may contain identifiable parts that can be arbitrarily complex descriptions. This resembles much more the object representations from description logics than the sparse objects that are used by the planning community. The rich structure of the message objects is essential to specify the flow logic and to provide mechanisms for message correlation.
- Finally, we remark that in contrast to classical planning, where all objects are available in the initial state and the actions change the state of objects, web services create new objects at runtime, i.e., they produce message objects during their execution that then can be further processed by other services. It is an open problem whether this behavior can be adequately modeled with the currently available planning techniques.

If one models Web services as action specifications similar to those used by AI planning, one can provide a more generic and powerful solution to the flow composition problem, i.e., build new flow plans from scratch (McDermott 2002; Srivastava 2002), but scalable algorithms to synthesize the required control structures are not easy to provide. Considering the BPEL4WS example, this would require to wrap the `receive` and `invoke` activities as the actions in the plan and specify their pre- and postconditions in an explicit way by referring to structural properties of incoming and outgoing messages and perhaps to the internal state of the BPEL4WS process. Note that no assumptions about the internal state of a partner process can be made as this information is not available. It is a fundamental assumption that partners can hide and constantly modify their processes behind the standardized message exchange interfaces as exposed by Web services.

A currently very relevant problem is the following: Given the message exchange behavior of one partner, i.e., the business protocol that he runs, construct a valid counterpart for another partner such that both can successfully communicate with each other. BPEL4WS is a very complex language for specifying business protocols and it is unknown how difficult the construction of a "dual" protocol is. From a planning perspective, it is challenging how to map this problem to the classical representation involving initial states and goals. A simpler approach to business protocol specification is the Web Services Conversation Language (WSCL) (Banerji & others 2002), which uses simple UML activity diagrams to specify a protocol. The 2-partner problem can be easily solved in WSCL, but the problem becomes difficult again for a multi-partner scenario. The problem is also discussed in (Piccinelli & others 2002) and a very preliminary solution is provided.

The decision-problem variant is also of highly practical relevance: Given two or more partner protocols, can these partners successfully communicate with each other? No results are available so far.

From this discussion, it becomes apparent that the web service composition problem is very similar to the design and specification of computer protocols, with the essential difference that the semantics of the new business protocol languages is not precisely defined. This major deficiency attracts more and more critics (Staab & others 2003). We see several possible candidates on which a formal semantics could be built: process algebras (Milner 1989), automata models (Brand & Zafiropulo 1983; Holzmann 1991), and the situation calculus (Reiter 1997) (recall the work in Golog that we discussed above). A related issue is how to give semantics to the input and output specification of a composite service where different outputs can be produced only when certain complex conditions are satisfied along some composition paths (Ankolenkar & others 2002).

The industry has high performance and availability requirements on Web services when they are deployed in a production environment. They want the Web services to be robust, meterable (so that service usage can be charged), secure (if needed), and verifiable. Some of these requirements are reflected in their specifications like BPEL4WS and there are initial attempts to verify them using formal methods (Fu, Bultan, & Su 2002). The Semantic Web has also started to look at the verification problem of flows. In (Narayanan & McIlraith 2002), the semantics of a subset of DAML-S is expressed in a first order language and Petri nets are used to simulate, compose, and verify Web services.

Related Work

The literature on Web services and the Semantic Web is abundant (Staab & others 2003) and the need for a more rigorous formal foundation is widely discussed. Currently, most of the work is in the description of Web services, the syntax of their flows, and how they could be executed. In the future, it is necessary to view Web services in the context of specifying, validating, and automatically synthesizing complex, reactive processes. Many areas of planning could become relevant in the future: the area of distributed planning (DesJardins *et al.* 1999), planning as model checking (Giunchiglia & Traverso 1999), or approaches that are between planning and the problem of synthesizing controllers (Barbeau, Kabanza, & St-Denis 1998), just to name a few. Automata-like representations have already been explored in a planning context (Dal-Lago, Pistore, & Traverso 2002), but so far they were focused on the representation of goals, not actions. In HTN planning (Erol, Hendler, & Nau 1994) operators can express aggregate behavior that can be further refined, but how to express nondeterminism and iterations in compound tasks is still an unexplored issue. Using planning for the travel domain has been demonstrated in (Ambite & others 2002), where an interactive framework of CSP solving is used to build travel plans in conjunction with inputs from the user. However, this approach does not involve the usage of nondeterministic or iterative operators. Planning has recently been explored for generating workflows in the Grid, but an explicit goal specification must be given (Blythe & others 2003). An application of regression planning to Web services is described in (McDermott 2002).

Conclusion and Future Work

Starting from the current interest in Web services, we explored the web service composition problem and compared the two major approaches to this problem—the industrial approach and the Semantic Web approach—with each other. We identified several, highly relevant subproblems and related them to the AI planning perspective. Although these problems resemble planning problems, it does not seem possible to directly apply current AI planning technology to them. We discussed the characteristics of these problems, which make them different from the commonly studied planning scenarios and we identified future research directions.

References

Ambite, J. L., et al. 2002. Getting from here to there: Interactive planning and agent execution for optimizing travel. In *Proc. IAAI*.

Ankolenkar, A., et al. 2002. DAML services. http://www.daml.org/services/.

Arkin, A., et al. 2002. Web services choreography interface WSCI. http://www.w3.org/TR/wsci/.

Banerji, A., et al. 2002. WSCL: The web services conversation language. http://www.w3.org/TR/wscl10/.

Barbeau, M.; Kabanza, F.; and St-Denis, R. 1998. A method for the synthesis of controllers to handle safety, liveness, and real-time constraints. *IEEE Transactions on Automatic Control* 43(22):1543–1559.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The semantic web. Scientific American, May issue.

Blythe, J., et al. 2003. The role of planning in grid computing. Proc. ICAPS.

Brand, D., and Zafiropulo, P. 1983. On communicating finite-state machines. *Journal of the ACM* 30(2):323–342.

Christensen, E.; Curbera, F.; Meredith, G.; and Weerawarana, S. 2001. The web services description language WSDL. http://www-4.ibm.com/ software/ solutions/ webservices/ resources.html.

Curbera, F., et al. 2002. Business process execution language for web services. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

Dal-Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In Dechter, R.; Kearns, M.; and Sutton, R., eds., *Proceedings of the 20th National Conference of the American Association for Artificial Intelligence*, 447–454. AAAI Press.

DesJardins, M.; Durfee, E.; Ortiz, C.; and Wolverton, M. 1999. A survey of research in distributed, continual planning. *AI Magazine* 20(4):13–22.

Erol, K.; Hendler, J.; and Nau, D. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In Hammond, K., ed., *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*, 249–254. AAAI Press, Menlo Park.

Fu, X.; Bultan, T.; and Su, J. 2002. Formal verification of e-services and workflows. Proc. ESSW.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In Biundo, S., ed., *Proceedings of the 5th European Conference on Planning*, LNAI. Springer.

Holzmann, G. 1991. *Design and Validation of Computer Protocols*. Prentice Hall, New Jersey.

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1-3):59–83.

McDermott, D. 2002. Estimated-regression planning for interactions with web services. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling*. AAAI Press, Menlo Park.

McIlraith, S., and Son, T. C. 2002. Adapting golog for composition of semantic web services. In Fensel, D.; McGuiness, D.; and Williams, M.-A., eds., *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, 482–493. Morgan Kaufmann, San Francisco.

Milner, R. 1989. *Communication and Concurrency*. Prentice Hall.

Narayanan, S., and McIlraith, S. 2002. Simulation, verification and automated composition of web services. In *Proceedings of the World Wide Web Conference*.

Piccinelli, G., et al. 2002. Web service interfaces for inter-organisational business processes - an infrastructure for automated reconciliation. In *Proc. EDOC*, 285–292.

RDF. 1999. RDF: Resource description framework. http://www.w3.org/RDF/.

Reiter, R. 1997. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. Preprint Version.

Srivastava, B. 2002. Automatic web services composition using planning. In *Proceedings of 3rd Internation Conference on Knowledge-Based Computer Systems*, 467–477.

Staab, S., et al. 2003. Web services: Been there, done that? *IEEE Intelligent Systems, Jan-Feb issue.* 72–85.