Planning for Requests against Web Services

Mike Papazoglou,[1,2] Marco Aiello,[1] Marco Pistore[1] and Jian Yang[2]

1. DIT, University of Trento
Via Sommarive, 14
38050 Povo, Trento, Italy

2. INFOLAB, University of Tilburg
PO Box 90153
NL-5000 LE Tilburg, The Netherlands

{mikep,jian}@uvt.nl {aiellom,pistore}@dit.unitn.it

Abstract

One of the most serious challenges that web service enabled e-marketplaces face is the lack of formal support for expressing service requests against UDDI-resident web services in order to solve a complex business problem. We present a framework in which such a formalization is possible by integrating AI planning and constraint satisfaction techniques with web service technology. This framework is designed to handle and execute requests performing planning under uncertainty on the basis of refinement and revision as new service-related information is accumulated (via interaction with the user and UDDI) and as execution circumstances necessitate change.

1 Introduction

The current phase of the e-business revolution is driven by enterprises that look to B2B solutions to improve communications and provide a fast and efficient method of transacting with one another. E-marketplaces are the vehicles that provide the desired B2B functionality. An e-marketplace is an electronic trading community that brings multiple customers, suppliers, distributors and commerce service providers in any geographical location together to conduct business with each other through the exchange of XML based messages (over the Internet) in order to produce value for end-customers and for each other.

Industry-based (or *vertical*) e-marketplaces, e.g., semiconductors, chemicals, travel industry, and aerospace, provide to their members a unified view of sets of products and services and enable them to transact business using diverse mechanisms, such as web services. The goal of web services when used within the context of e-marketplaces is to enable business solutions by assembling and programming pre-built software components offering business functionality on the Web. Each of these components behaves like a self-contained, modular mini-application with its own interface described in the web service description language WSDL (`www.w3.org/TR/wsdl`) that can be published and invoked over the Internet. This allows companies to conduct electronic business, by invoking web services, with all partners in a marketplace rather than with just the ones with whom they have collaborative business agreements. Service offers are described in such a way, e.g., WSDL over UDDI (`www.uddi.org`), that they allow automated discovery to take place and offer request matching on functional and non-functional service capabilities.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

One of the biggest challenges that web service enabled e-marketplaces face is the lack of support for appropriate service request languages that retrieve and aggregate services which contribute to the solution of a business problem. Users typically require services from an e-marketplace based on their characteristics and functionality. Currently, there are no formal specification mechanisms for formulating user requests and no automated support for expressing requests over UDDI-resident services other than the primitive enquiry portion of the UDDI API. Requests based on the UDDI enquiry API are programmed within application programs and need serious re-coding efforts every time that there is need for a new request or an extension to a previous request. The baseline desiderata for a request language, at a much higher level of abstraction than what is currently offered by primitive UDDI APIs, are the following:

- *Genericity* The language must be generic in that it can be used for the same kind of services offered by different e-marketplaces.
- *Separation of goals* The language should distinguish between qualitative and quantitative goals. The former are more abstract general goals, e.g., a business process goal, the latter are the quantitative values tied to such goals, e.g., achieving a particular business process goal within a specified time-frame.
- *Reactiveness to non-determinism and change* The language must be able to express different sub-goals depending on events which are not known a priori and are not under the control of the requester, e.g., the requester does not know beforehand and cannot control whether a payment is going to be accepted, or the conditions of a particular instance of a web service.
- *Interoperability* The constructs of the request language should be directly mappable to existing web service standards and protocols. In other words, it should be possible to translate a request into a program interacting with web services as specified by current standards.

There are also a number of features, which would further enhance the usability and expressive power of the language. These include the following items:

1. The language could be enriched with the capability of distinguishing between different types of goals and sequencing preferences. A user needs to be able to state that one goal is preferable to another one, which in turn is preferable to a third one. For example, it should be possible to state that a goal is vital or that it is optional, and also express an explicit ordering of goals based on personal preferences.
2. The language could be enriched with similarity operators. There are three levels at which such semantic operators can work: **(a)** at the constraint objects level: a user could specify that s/he wants something similar to a compact car (topology), or some location close to a given city (proximity), and so on; **(b)** at the service level: a user could specify that s/he wants something functionally similar to a specific service, e.g., we may choose to replace a train service by a plane service; **(c)** at the plan level: a user could specify a goal similar to an already successfully executed plan, e.g., make a trip similar to the one the user has previously requested or done.

Our research concentrates on creating such a language and ensure an appropriate execution environment for it. For this purpose we do not rely solely on web service technologies, but also use AI planning techniques, which serve the purpose of expressing high-level goals and the handling of non-determinism and change, and constraint satisfaction techniques to ensure the satisfaction of the quantitative part of user requests. In [1] we introduced the basic constructs of this kind of language by integrating temporal constructs for expressing goal sequencing and constraints over quantitative values over the goals, in [2] we have moved one step further by completely specifying a request language, called XSRL, and described an execution environment for it. In this paper, we present the main constructs of XSRL, briefly overview its execution environment and give an example of an XSRL-based request.

2 Expressing requests against web services

In the following we use an application scenario in the business domain of e-travelling based on the specifications of the open travel agency (OTA, `www.opentravel.org`), where we consider an application booking flight segments and making hotel reservations for travellers. OTA has specified a set of standard business processes for searching for availability and booking a reservation in the airline, hotel and car rental industry, as well as the purchase of travel insurance in conjunction with these services. OTA specifications use XML for structured data messages to be exchanged over the Internet.

For the purposes of this paper we have chosen the OTA schema (document model) for an air segment reservation. This business process specifies the format of an air segment schema in XML (WSDL and BPEL) can be used as alternative representations) as well as the request and response formats for the air segment schema. The input (request) document necessary for the air flight segment includes departure and arrival airports, arrival and departure dates, desirable price ranges and seat numbers. The output document may include similar information but with actual destinations, dates and prices that are supplied after interacting with service providers. A comparable schema exists for hotel reservation purposes.

In Figure 1, we use an activity diagram to represent graphically the standard OTA air segment reservation business process. Formalising the OTA specification with an activity diagram is just one of many possible options, which is currently only used for illustration purposes. Solid arrows in the diagram represent flows of control while dashed arrows represent flow of messages. Each node in the diagram represents an activity. This formalism allows for cycles shown as outgoing arrows from end states. There are two business processes in this activity diagram, a travel agent business process and a tour operator business process. Business processes specify how agents, e.g., a booking application (user application in Figure 1), a travel agent, and a tour operator, in an e-travelling marketplace interact in order to satisfy a traveller's requests. The interaction between roles takes place as a choreographed set of business steps or actions that can be represented in BPEL.

In Figure 1, the business process is initiated by a traveller who provides the OTA request document necessary for an air flight segment (arrival airports, dates, seat quantity, etc). This triggers a request activity at some travel agent, which in turn triggers a package availability request for the from some tour operator. Air segment reservation information messages are exchanged until the traveller is able to choose between rejecting, modifying or accepting the air segment offered by the service providers. Potential users (travellers) can come up with requests to book their holidays on the basis of the business process described by this activity diagram. A similar process can be used for hotel reservation purposes but is not illustrated due to space limitations.

The request shown in Figure 2 expresses the wishes of a user who wants to travel from New York to a destination in Italy such as Rome or Venice. This traveler wants to use Alitalia or United Airlines as flight carrier, spend between 500 and 800 dollars per passenger, reserve 3 seats on the flight, leave on the 1st of June and return on the 10th. These are the input parameters required by the standard XML schemas and business process specification of the OTA marketplace. The traveler also requires accommodation for the same destination, in an hotel of the Hilton chain. Furthermore, the traveler finds it vital to have a flight ticket, but would still travel even if s/he did not acquire a hotel reservation. Obviously, the passenger does not wish to reserve accommodation without a confirmed flight ticket. These wishes are expressed in XSRL using the code snippet depicted in Figure 2 and are executed against the canonical air segment reservation process schema, which is represented graphically in Figure 1.

The interpretation of the request in Figure 2 follows. The "pure" request is specified in the part enclosed between the `<REQUEST>` tags. In the part enclosed between the `<GOAL>` tags the user specifies that s/he wishes to receive a confirmation with respect to a variable `$a`, and this goal is `<vital>` with respect to the request. If this succeeds `<then>` he is also willing to receive a confirmation for an hotel `$h`, but this is an `<optional>` goal (that is, if it fails, the whole request should still proceed). The portion of the request enclosed in the `<REQUEST>` tags defines the quantitative values tied to the various variables named in the goal portion and also specified the way the information is returned to the requester in case of success.

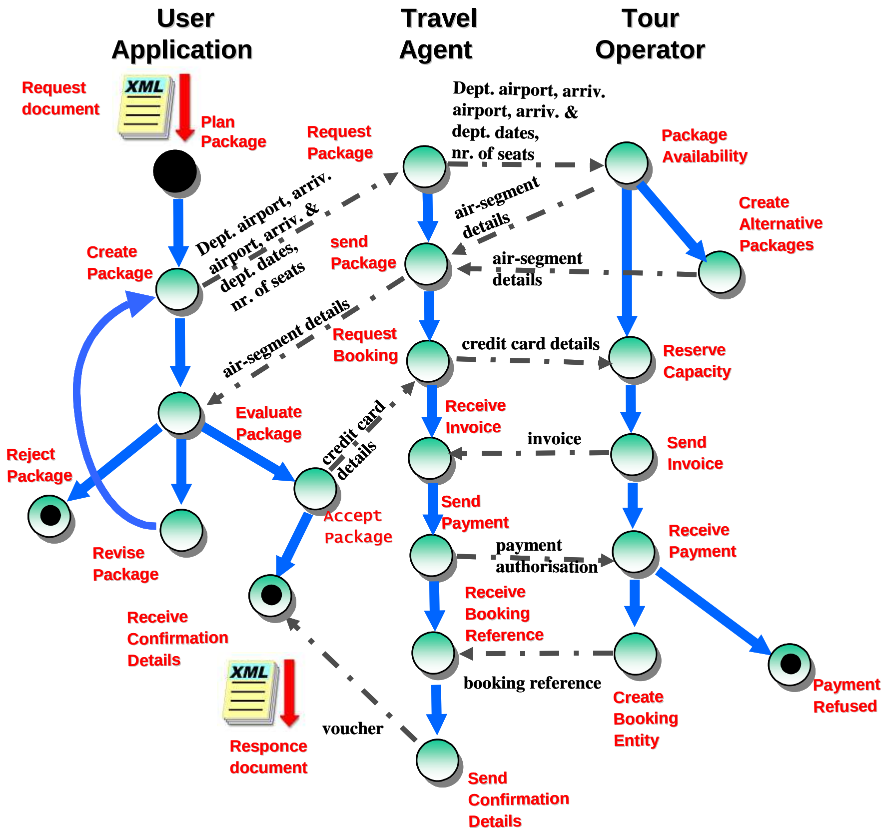

Figure 1: The AirSegment activity diagram.

3 The planning framework

The XSRL language has the properties of being generic, separating goals, dealing with change and non-determinism and of interoperability. We illustrate how these characteristics are achieved by briefly sketching the execution environment underlying the XSRL. The language is built on two inter-dependent components:

1. A "pure" *request specification component* that enables users to express a request in terms of the language constructs and enables complex formulations over these constructs,

2. and a *scheduling or goal component* that expresses user objectives (goals) as well as scheduling preferences and dependencies among the requested services.

The request specification component is specified in terms of XQuery constructs (`www.w3.org/TR/xquery`) and is combined with the scheduling part that is developed on the basis of the planning language $EaGLe$ [3], which it also extends. The idea is that one can describe activities and goals and compose them, and decide their sequencing. In addition, one can also use constructs that react to failure and non-determinism, such as "try to achieve a goal" or "if a subgoal fails then try to achieve an alternative subgoal." The language $EaGLe$ is fully implemented in a working prototype which comprises a planner MBP (model based planner). To work, the MBP needs an $EaGLe$ request expressed against a domain model. The domain model in the case of XSRL is a set of standard business processes. The descriptions of such processes are usually available in the case of web services in form of definitions based on modelling and specification languages such as BPEL (`www-106.ibm.com/developerworks/webservices/library/ws-bpel`) or BPML (`bpmi.org`). These need to

```
<XSRL>
   <REQUEST> {
      FOR $a in document(PkgTravelSegment.xml)//AirSegment
         [CarrierName =  "Alitlaia"| "United Airlines" AND
          DepartureAirport = "NewYork" AND
          ArrivalAirport = "Rome" | "Venice" AND
          (Price <= 800 AND Price >=500) AND
          SeatQty = 3 AND
          ArrivalDate = "1 June, 2002" AND
          DepartureDate = "10 June, 2002"]
      RETURN
         <ArrivalAirport>{ $a/ArrivalAirport}</ArrivalAirport>
         <price>{ $a/price}</price>
         <ArrivalDate>{ $a/ArrivalDate}</ArrivalDate>
         <DepartureDate>{ $a/DepartureDate}</DepartureDate>
         <HotelList> {
            FOR $h in document (hotelReference.xml)//HotelReference
               [ChainHotel = "Hilton"]
            WHERE ($h/Area =$a/ArrivalAirport AND
                   $h/HotelArrivalDate = $a/ArrivalDate + 1 AND
                   $h/HotelDepartureDate = $a/DepartureDate 1)
            RETURN
               <HotelName>{ $h/HotelName }</HotelName>
               <HotelAddress>{ $h/HotelAddress }</HotelAddress>
         }</HotelList>
   }</REQUEST>
   <GOAL>
      <Then><Vital>receive_confirmation($a)</Vital>
      <Optional> receive_confirmation ($h)</Optional></Then>
   </GOAL>
</XSRL>
```

Figure 2: An XSRL request.

be translated into a format understandable by the MBP planner ($NuPDDL$, a non deterministic extension of the Planning Domain Description Language, [4]). The output of MBP after a request is a program which represents the plan corresponding to the request of the user. This plan, named *generic plan*, is also encoded as an XML document.

The user specifies quantitative properties related to the business activities he desires to engage in. These are invoked via the UDDI registry in order to qualify the service requests. The responses of the web services contacted via the UDDI and the values provided in the request need to be matched. This is done by a constraint satisfaction solver, which takes all the typed numeric values and propagates the constraints. The result is a set of satisfied constraints, which are tied to a request and to a particular web service that partially satisfies it. These are then bind to the generic plan, yielding a set of plans that can potentially achieve the goals expressed in the request. We call a plan belonging to this set an *instantiated plan*.

Finally, the requester can inspect and change the instantiated plans via iXSRL: a set of interactive XSRL constructs that express acceptance, revision or rejection of instantiated plans. We refer the reader to [2] for further details.

Figure 3 provides a high-level view of the architecture for interpreting and executing XSRL requests. In this figure the user issues an XSRL request which is divided into its basic components. The goal portion is send to the MBP planner, which also needs a business process description. MBP generates a generic plan that interacts with the UDDI API to retrieve services from the UDDI registry. The values returned by web services offered by the UDDI registered providers are then matched with the constraints provided by the user via the XSRL request. In this manner we obtain a set of instantiated plans which are managed by the user via an iXSRL expression.

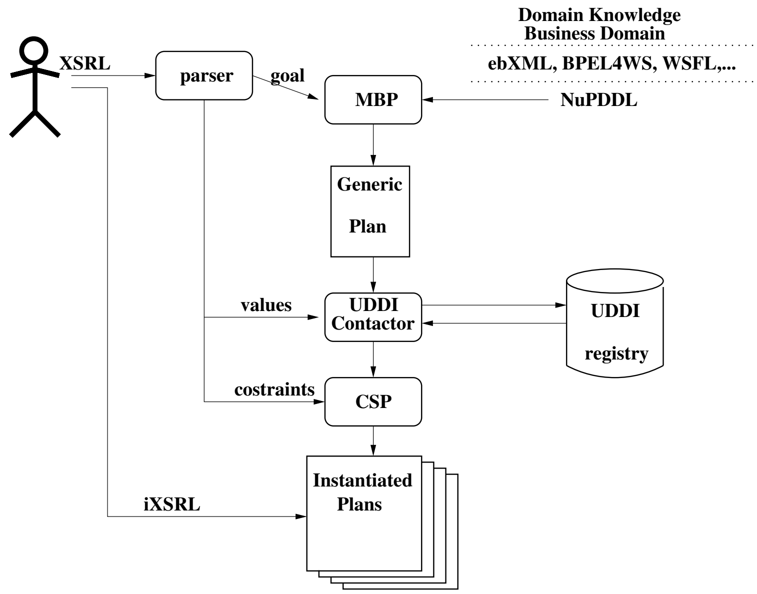

Figure 3: The XSRL request execution framework.

4 Concluding remarks

In this paper we highlighted an approach for web service interaction based on planning and constraint satisfaction and a service request language developed on the basis of this framework. The planning framework was developed on the basis of a coherent view of the issues arising when planning requests against web services under uncertainty (as plans inevitably do not execute as expected) in dynamic environments where there is the constant need to be able to identify critical decision trade-offs, revise goals and evaluate alternative options. This approach recognises that in an uncertain and dynamic world such as that of web services a correspondence must be drawn between the formal representation of a business domain model and the planer's model of it. It then instantiates plans on the basis of the plan model in terms of a user specific request and via interaction with the UDDI infrastructure; and when necessary, dynamically reconfigures plans on the basis of user interaction. These design considerations are reflected at the level of the request language that generates plans over web services residing in an e-marketplace and its run-time environment.

References

[1] M. Aiello, M. Papazoglou, J. Yang, M. Carman, M. Pistore, L. Serafini, and P. Traverso. A request language for web-services based on planning and constraint satisfaction. In *VLDB Workshop on Technologies for E-Services (TES02)*, 2002.

[2] M. Papazoglou, M. Aiello, M. Pistore, and J. Yang. XSRL: An XML web-service request language. Technical Report DIT-02-0079, Univ. of Trento, 2002.

[3] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *18th National Conference on Artificial Intelligence (AAAI-02)*, 2002.

[4] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language. In R. Simmons, M. Veloso, and S. Smith, editors, *4th Int. Conf. on AI Planning and Scheduling*, 1998.