UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

XSRL: AN XML WEB-SERVICES REQUEST LANGUAGE

Mike Papazoglou, Marco Aiello,
Marco Pistore and Jian Yang

2002

Technical Report # DIT-02-0079

XSRL: An XML web-services request language

Mike Papazoglou{1,2}, Marco Aiello[1], Marco Pistore[3] and Jian Yang[2]

1. DIT — Univ. of Trento, Via Sommarive, 14, 38050 Trento, Italy
email: aiellom@dit.unitn.it, `http://www.dit.unitn.it/~aiellom`

2. INFOLAB — Tilburg Univ., PO Box 90153, NL-5000 LE Tilburg, The Netherlands
email: {mikep,jian}@kub.nl, `http://infolab.kub.nl/people/{mikep,jian}`

3. ITC-IRST, Via Sommarive,18, 38050 Trento, Italy
email: pistore@irst.itc.it, `http://sra.itc.it/people/pistore`

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.

In this paper we present a web-service request language (XSRL) developed on the basis of AI planning and the XML database query language XQuery. This framework is designed to handle and execute XSRL requests and is capable of performing planning actions under uncertainty on the basis of refinement and revision as new service-related information is accumulated (via interaction with the user or 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, aerospace, etc, 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 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, that they allow automated discovery to take place and offer request matching on functional and non-functional service capabilities.

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 that contribute to the solution of a business problem. Users typically require services from an e-marketplace based on their characteristics and functionality. A service request language provides for a formal means of describing desired service attributes and functionality, including temporal and non-temporal constraints between services, and service scheduling preferences. 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 research presented herein concentrates on developing a service request language for XML-based web-services in e-marketplaces that contains a set of appropriate constructs for expressing requests and constraints over requests as well as scheduling operators. We name this language XSRL for **X**ML **S**ervice **R**equest **L**anguage. XSRL expresses a request over web-services and returns a set of documents as the result of the request, e.g., by constructing an end-to-end holiday package (documents) comprising a number of optimised flight and accommodation choices. Loosely speaking, the response documents can be perceived as a series of plans that potentially satisfy a request. In expressing an XSRL request it is important that a user is enabled to specify the way that the request needs to be planned and executed. Hence, it is our conviction that a service request language, such as XSRL, should comprise two inter-dependent components:

1. A “pure” *request specification component* that enables users to develop definitions that express in formal terms the core entities of a request and enables complex formulations over these core entities,

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

A request written in XSRL is input to an automated planner/scheduler. After receiving the requests from a user, the planner generates a schedule for interacting with the service providers without further interaction from the user. The request language allows the planner to select suitable initial and final actions for requested service combinations and plan its actions so that it meets the original constraints of the user. Accordingly, a request expressed in XSRL results in the generation of executable plans describing both the sequence of plan actions to be carried out in order to satisfy the user request and all necessary information and constraints essential to develop each planned action in correct and consistent manner.

The field of AI planning offers high potential for solving problems by instilling web-based planning and scheduling capabilities into distributed decision-making agents in order to manage resource-constrained domains [1]. Recently, different planning approaches have been proposed (see [2, 3, 4]), most of which focus on gathering information from the web and on applying deterministic planning. In our view, AI planning provides core facilities for developing a web-services request language and for checking the correctness of the plans that are generated by it.

Our research todate has concentrated on combining a goal language for expressing extended goals in non-deterministic domains [5] with a system-level planning language for interacting with web services [6]. In [7] we sketched a high-level architectural view and the initial language support requirements of a system allowing to express requests over UDDI-resident web-services. Our contribution in this paper focuses in request languages for web-services and concentrates on introducing high-level lightweight language constructs for XSRL. Thus, we have chosen to: (1) use an AI planner for non-deterministic domains, such as, for instance open travel; and (2) work with a combination of an extended temporal language and XQuery constructs for expressing requests against UDDI-resident web-services. XSRL can express information both about the request, the succession of activities needed to satisfy a request and over the parameters of the planned actions. In addition, the plans generated by the language are capable of dealing with non-determinism, interleaving and constraint satisfaction.

This paper is organised as follows: in Section 2, we discuss the importance and requirements for modelling a business domain and the use of activity diagrams for this purpose. Section 3 presents a conceptual view of the architectural framework for supporting requests over UDDI-resident services. In Section 4, we introduce a service request language (XSRL) for web-services, describe its syntax and formal semantics, and explain how XSRL requests are executed and implemented. In Section 5, we discuss related work, while in Section 6 we present our summary and conclusions.

2 Modelling the business domain

In order for enterprises to conduct electronic business with each other in an e-marketplace, they must first discover each other and the products and services they have to offer. Subsequently, they must determine which business processes and documents are necessary to obtain those products and services. Finally, they need to determine how the exchange of information will take place and then agree on contractual terms and conditions. Once all of these activities are accomplished, then they can exchange information and products/services according to these agreements.

To facilitate this endeavour, vertical e-marketplaces provide an infrastructure for efficiently managing interactions between buyers and sellers. A vertical e-marketplace provides data and business process definitions including relationships and cross-references and a business terminology organized on a per industry domain basis. For example, business standards such as RosettaNet and ebXML describe business processes, as well as e-business interactions that should be carried out so that transactions can be executed across organisations by exchanging business documents, such as purchase orders, invoices, tender requests and so on.

For each *domain model* specified by a vertical e-marketplace we require at least three elements in order to enable automated e-business exchanges and application development:

1. Standard business processes that formally describe business interactions between organisations. In the world of web-services business processes are published via e-business directories such as the UDDI. For example, ebXML uses the UN/FECACT Meta Model (UMM) as a mechanism to allow trading partners to capture the details for a specific business scenario using a consistent modelling methodology.

2. A document model for defining structured XML business documents exchanged between

trading partners or service requesters and providers over the Internet. This includes request (input) and reply (output) business documents for business actions that are part of standard business processes.

3. A business lexicon that provides standard business terminology definitions including relationships and cross-references as expressed in the business terminology and organized by industry domain.

```
?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace=http://www.opentravel.org/OTA
 xmlns="http://www.opentravel.org/OTA"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
    <xs:include schemaLocation="PkgCommonElements.xsd"/>
       <xs:complexType name="PkgTravelSegment">
           <xs:annotation>
              <xs:documentation> A full definition of a travel segment.
              </xs:documentation>
           </xs:annotation>
           <xs:choice>
               <xs:element name="AirSegment" type="PkgAirSegmentType"/>
               <xs:element name="FerrySegment"type="PkgFerrySegmentType"/>
           </xs:choice>
       </xs:complexType>
       <xs:complexType name="PkgAirSegmentType">
           <xs:sequence>
               <xs:element name="DepartureAirport"
                   type="PkgTravelLocationName" minOccurs="0"/>
               <xs:element name="ArrivalAirport"
                   type="PkgTravelLocationName" minOccurs="0"/>
               <xs:element name="AvailableSeats" minOccurs="0">
                          ......  ......
               </xs:element>
            </xs:sequence>
            <xs:attribute name="Duration" type="xs:string">
            </xs:attribute>
            <xs:attribute name="CarrierCode" type="xs:string"
                          use="optional">
            </xs:attribute>
            <xs:attribute name="CarrierName" type="xs:string"
                          use="optional">
            </xs:attribute>
                          ......  ......
</xs:schema>
```

Figure 1: An excerpt of the AirSegment XML schema.

In this paper, we base our application scenario on the business domain of e-travelling where we consider an application booking flight segments and making hotel reservations for travellers. We base our example on the specifications of the open travel agency [8]. OTA has specified a set of standard business processes for availability searching and reservation booking 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.

An excerpt from the OTA schema (document model) for an air segment reservation is illustrated in Figure 1. This business process specifies the format of an air segment schema in XML as well as the request and response formats for the air segment schema in an open-travel marketplace that offers web-service functionality. 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 with actual destinations, dates and prices that are supplied after interacting with service providers. A comparable schema exists for hotel reservation purposes. This example was chosen for illustrative purposes and for ease of understanding.

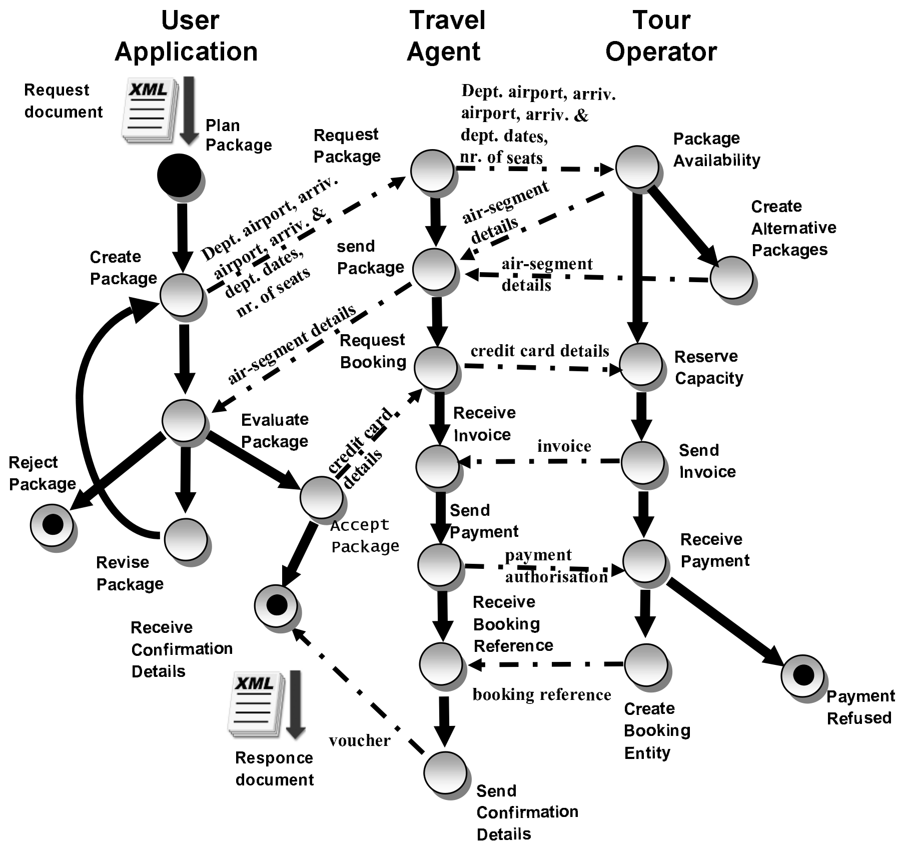

Figure 2: The AirSegment activity diagram.

In Figure 2, we use an activity diagram as an aid to formally represent business process specifications. In particular, this figure represents a standard air segment reservation business process. Solid arrows in this diagram represent flows of control while dashed arrows represent flow of messages. Each node in the diagram represents an activity. Following the UML convention, we use solid filled circles to denote initial states and circles surrounding a small solid circle to denote end states. 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 2), 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. Each business action is expressed as an exchange of data in the form of XML electronic business documents.

In Figure 2, the business process is initiated by a traveller who provides the request document necessary for an air flight segment (arrival airports, dates, seat quantity, as shown in the input document in Figure 1). This triggers a request activity at some travel agent, which in turn triggers a package availability request 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.

Formalising the OTA specification with an activity diagram is just one of many possible options, which is used for illustration purposes only. Here we are not concerned with which is the most expressive formalism for modelling web-service interactions, rather we are interested in presenting an architecture for interaction with e-marketplace based web-services that is independent from the chosen business modelling formalism.

3 Conceptual view of the planning framework

There are a number of requirements that a web-services centred planner should satisfy. The plans should have an "open dynamic structure" and be situated in the context of execution. This means that plans should deal with non-deterministic behaviour in the business domain and the set of potentially executable actions may change during the course of planning. For instance, once a travel plan has been generated and proposed to the user, the user may decide to change some of the parameters connected to an action, e.g., hotel booking preferences and parameters, by dynamically reconfiguring the plan. Hence, plans should be amenable to refinement and revision as new information is accumulated (via interaction with the user or UDDI) and as execution circumstances necessitate change. As plans inevitably do not execute as expected there is the constant need to be able to identify critical constraints and decision trade-offs, and for evaluation of alternative options. In the planning parlance, this means that it should be possible to deal with interleaved plans, interactive plans, reactive plans that deal with exogenous events, e.g., information supplied by the UDDI, and contingency plans that deal with uncertain outcomes of non-deterministic actions. The need for dynamic planning capabilities is at direct odds with classical planning research that assumes complete knowledge of the conditions in which the plan will be executed, deterministic execution and actions with predictable outcomes and little space for incremental changes. In addition to these requirements, there is also the core requisite of being able to prove the correctness of the correspondence between a request and the plans generated by the request language as a response to this request.

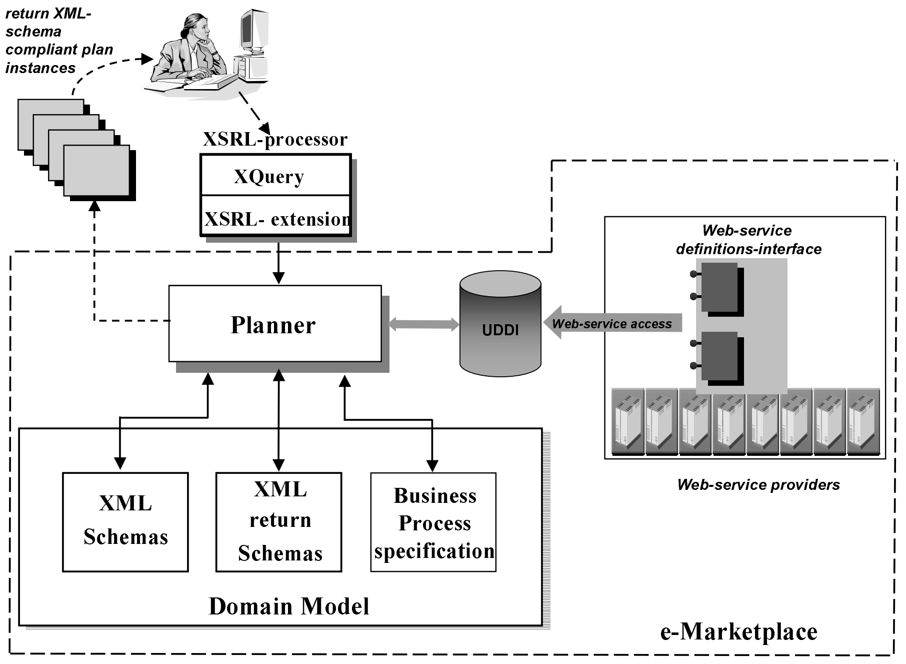

Figure 3: The E-Marketplace architecture.

Figure 3 presents a conceptual view of the planning framework for web-services. As shown in this figure, a user submits a request expressed in XSRL that is received by the planner. Subsequently, the planner plans a series of actions that need to be executed on the basis of information supplied by the domain model. The domain model contains information such as the XML schemas for the business processes in the domain, e.g., air segment reservation in Figure 1, as well as the request and response formats for these XML schemas. The actions that are planned are based on a business process specification for formally describing the actions within and between business processes. More specifically, in the context of our running example, the planner plans the actions of the air flight segment booking (user) application Figure 2, e.g., `create-package`, `evaluate-package`, `receive-package`, and so on. It also plans their interactions with actions that are external to the user application, e.g., `request-package`, `send-package`, etc. Other actors in the business model such as the TravelAgent and TourOperator support these external actions. Actions internal to the business processes provided by the TravelAgent and TourOperator are the responsibility of these service-providers and are external to the planner. The planner makes only certain that business process interfaces and actions of user applications and those provided by service providers are conformant and consistent with respect to one another.

Once the planner has generated an initial plan of how to satisfy a request expressed in XSRL, it interacts with UDDI-based service providers and their web-services. The planner generates a series of XML documents in response to plans executed on the basis of data supplied by web-service operations offered by UDDI resident service-providers and which are compliant with the XML input schema. These XML documents are passed to the user for inspection and include return schema compliant data, such as arrival airports, arrival and departure dates, and price ranges.

In this paper we focus on the service request part of the planning framework and describe its syntax and formal semantics on the basis of our running example.

4 The request language XSRL

When users interact with e-marketplace based web-services, ideally they wish to specify only their goals and to obtain the desired results. Consequently, the basic premise of our approach is to equip the user with an expressive service request language so that s/he can specify his objectives clearly and unambiguously. The language should be lightweight and comprehensive. In other words, it should be possible to state a request in as much or as little detail as the case may require. Key capability for a service request language would be to satisfy a user request by transparently assembling service solutions from different service providers in the e-marketplace dynamically by aggregating and composing services, e.g., by constructing an end-to-end holiday package comprising a number of optimised flight and accommodation choices.

4.1 The request language

The services request language XSRL is partly based on the XQuery language and extends it appropriately with constructs from a formal AI planning language, called *EaGLe* [5], that is used for expressing extended goals.

The basic form of a request expressed in XSRL is:

```
request ::= ‘<XSRL>’ xquery_expression goal_formula ‘</XSRL>’
```

The results of a request formulated in XSRL are presented in an XML document as specified in the XQuery part of the request.

XQuery is a powerful typed functional language [9], capable of selecting and extracting complex patterns from XML documents and of reformulating them into results in arbitrary ways. An XQuery statement is built from expressions (called queries) which can be composed arbitrarily. Expressions can be nested and variables are always passed by value. Types can be imported form XML schemas and XQuery can then perform operations based on these types. The BNF definition of the `xquey_expression` inside an XSRL request follows:

```
xquery_expression ::= ‘<XQuery>’ FlwrExpr ‘</XQuery>’
```

XQuery introduces FLWR subexpressions that contain: FOR, LET, WHERE, and RETURN statements. The BNF productions in the XQuery grammar that define a FLWR expression are as follows [9]:

```
FlwrExpr      ::=  (ForClause | letClause)+ whereClause?  returnClause
ForClause     ::=  ‘FOR’ Binding_Var ‘IN’ Expr
                   (‘,’ Binding_Var ‘IN’ Expr)*
LetClause     ::=  ‘LET’ Binding_Var ‘::=’
                   Expr (‘,’ Binding_Var ‘::=’ Expr)*
WhereClause   ::=  ‘WHERE’ Expr
ReturnClause  ::=  ‘RETURN’ Expr
```

An FLWR statement consists of a series of one or more FOR and/or LET clauses, followed by an optional WHERE and terminated by a mandatory RETURN.

The *EaGLe* language is closely related to the known temporal logic CTL [10]. It inherits from CTL the possibility of declaring properties on the non-deterministic temporal evolutions of a system, and extends CTL with constructs that are useful for expressing complex planning goals, such as sequencing of goals, and goal preferences. The BNF definition of the `goal_formula` is an extension of *EaGLe* with XML and service-related constructs. Its purpose is to allow users to associate intensions (goals) with a service request. The `goal_formula` is expressed as follows in terms of BNF:

```
goal_formula  ::=  '<Goal>' goal '</Goal>'
goal          ::=  basic_goal | '<And>' goal goal '</And>'
                   '<Then>' goal goal '</Then>' |
                   '<Fail>' goal goal '</Fail>' |
                   '<Repeat>' goal '</Repeat>'|
                   '<Vital>' basic_goal '</Vital>' |
                   '<Optional>' basic_goal '</Optional>' |
                   '<MaintainVital>' property '</MantainVital>' |
                   '<MaintainOptional>' property '</MaintainOptional>'
basic_goal    ::=  Process ( '(' Binding_Var ( ',' Binding_Var )* ')' )?  |
                   '<BooleanNot>' basic_goal '</BooleanNot>'
                   '<BooleanOr>' basic_goal basic_goal '</BooleanOr>' |
                   '<BooleanAnd>' basic_goal basic_goal '</BooleanAnd>'
property      ::=  Constraint ( '(' Binding_Var ( ',' Binding_Var )* ')' )?  |
                   '<BooleanNot>' property '</BooleanNot>'
                   '<BooleanOr>' property property '</BooleanOr>' |
                   '<BooleanAnd>' property property '</BooleanAnd>'
```

where the `Binding_Var` expression is an XQuery variable name and `Process` is the name of a standard process found in the e-marketplace business process specification. Basic goals, defined by the `basic_goal` production rule, can be simple `Process (Binding_Var, Binding_Var,...)` expressions or Boolean compositions. Vital and optional statements over basic goals and maintaining statements over properties form `goals`, which can be combined in more complex goals with `And, Then, Fail, Repeat` statements. This is defined by the `goal` production rule.

XSRL allows for expressing conditions that are *vital* to reach (e.g., <Vital> a($v) </Vital>) or to maintain (e.g., <MaintainVital> c($v) </MaintainVital>), as well as conditions that are *optional* or *desired* (<Optional> a($v)</Optional> or <MaintainOptional> c($v) </MaintainOptional>),

```
<XSRL>
   <XQUERY> {
      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>{ $h/HotelAddress }</HotelAddress>
         }</HotelList>
   }</XQUERY>
   <GOAL>
      <Then><Vital>receive_confirmation($a)</Vital>
      <Optional> receive_confirmation ($h)</Optional></Then>
   </GOAL>
</XSRL>
```

Figure 4: An XSRL request.

where a($v) signifies a certain process and c($v) denotes a constraint over a binding variable. Moreover, it allows for expressing *concatenation* of goals. For example, the expression <Then> <BooleanOr>a_1($v) a_2($v′) </BooleanOr> a_3($v″)</Then> states the fact that process a_3($v″) has to occur after at least one of the processes a_1($v) or a_2($v′) has happened. It also allows expressing *preferences* between goals. For example, the expression <Fail> <Optional> a_1($v) </Optional> <Vital> a_2($v′) </Vital> </Fail> states the fact that the preferred goal is to achieve process a_1($v), however if that is not possible, then it is vital to achieve at least a_2($v′). We refer to [5] for a complete description and for the full semantics of the *EaGLe* language.

XSRL is quite a powerful language to express requests against complex web services or compositions of web services. In the following, we introduce a simple XSRL request on the basis of the XML schemas for the definition of a travel segment (`http://www.opentravel.org/OTA/PackageTravelSegment.xsd`) and for the hotel references (`http://www.opentravel.org/OTA/HotelReference.xsd`) which are part of the running example introduced in Section 3.

The code snippet in Figure 4 is the request of a traveller who wishes to travel from New York to a destination in Italy such as Rome or Venice. This traveller wants to use Alitalia or United Airlines as flight carrier, spend between 500 and 800 dollars, 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 traveller also requires accommodation for the same destination, in an hotel of the Hilton chain. Furthermore, the traveller finds it vital to have a flight ticket, but would still travel even if s/he did not have a hotel reservation. Obviously, the passenger does not wish to reserve accommodation without a confirmed flight ticket.

The XQuery part of the request in Figure 4 is rather straightforward. However the goal part deserves some explanation. The goal is subdivided into two subgoals to be achieved sequentially; this is captured by the '<Then>' statement. The first subgoal is considered to be vital, therefore the '<Vital>' statement is used. The activity in this statement is that of receiving a confirmation for the part of the request designated by the air travel segment variable \$a. The second subgoal is optional and is indicated by the '<Optional>' statement. The action in this statement is to receive a confirmation for the request designated by the hotel variable \$h. In both cases, the receive confirmation action is the final action in its respective business process. This indicates that the AirSegment and Hotel reference business processes must complete. This can only happen if the action `receive_confirmation($a)` is successful for the AirSegment and the action `receive_confirmation($h)` is successful for the Hotel business process.

The above example has illustrated some of the main construct of XSRL. XSRL can be used to code fairly complicated examples involving web service interactions, synchronization sequences, and temporal constraints over interacting web-services.

4.2 Execution of an XSRL request

An XSRL request needs a number of interacting building blocks to be executed. The planning framework presented in Figure 3 is responsible for executing the various tasks associated with an XSRL request: interpreting the planning instructions in the goal expression, interacting with UDDI, enforcing the user's constraints and interacting with the user in order to allow him to choose among alternative solutions and to refine the request. These tasks are illustrated in Figure 5. This figure gives a high level view of the execution phases of an XSRL request.

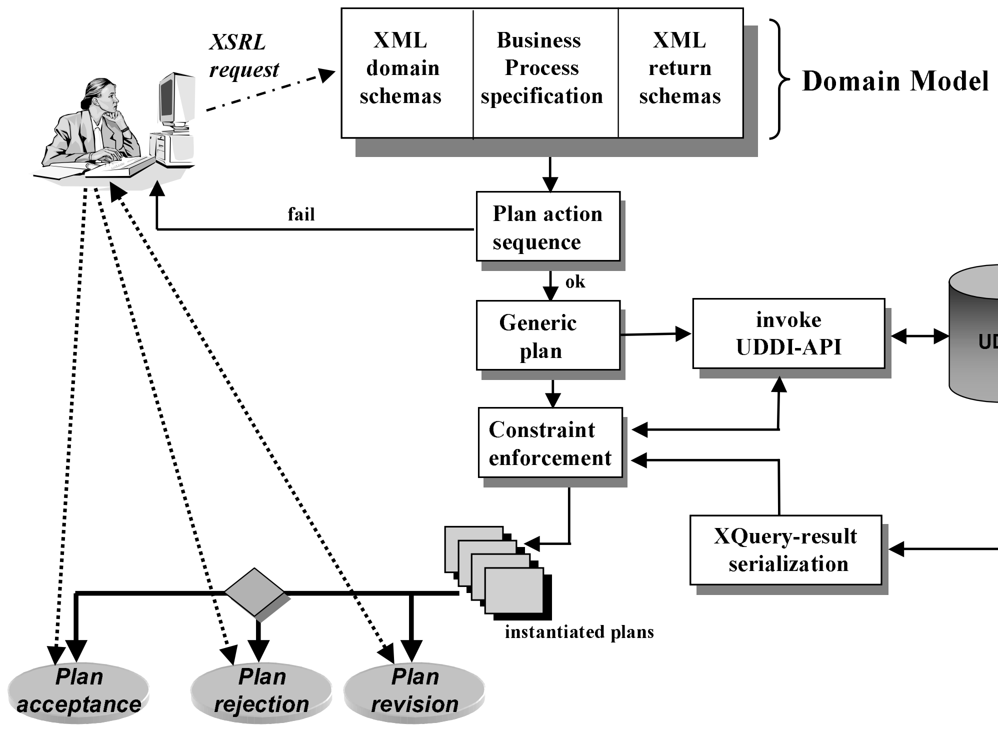

Figure 5: Execution steps of an XSRL request.

The process of executing an XSRL request is initiated by a user who is aware of the format of the input and the output parameters as defined in the XML schemas and the business process specifications for a particular domain. These are part of the domain model available to the planner as illustrated in Figure 5. The purpose of these XML schemas and business process specification is to assist the user in forming a correct XSRL request. Note that the XSRL request includes also the manner in which the output is presented. This capability is inherited from the XQuery portion of the language and this is the reason why the user needs to also consult the XML return schemas. The planner module in Figure 5 is a Model Based Planner (MBP) [11] that interprets the goal portion of the XSRL request and identifies the sequence of activities that need to be performed in order to achieve the specified goal traversing business process paths. The planner checks the consistency of the request with respect to the business process specification by finding appropriate paths of activities leading to the final activity specified in the goal part of XSRL. Intuitively, the paths are computed by executing the request against a business process specification, e.g., an activity diagram such as the one illustrated in Figure 2. If the request is consistent with the domain model specifications, it returns a generic plan for further processing. If the request is inconsistent, then the user is prompted for reformulation of the request. Note that the planner returns activity paths along the business process specification that could potentially satisfy the request. However, at this stage the planner is not concerned with the values of the schema elements and attributes tied to the actions in the business process, e.g., actual ticket prices, destinations and so on. We refer to this type of plan as a *generic plan.* An example of a generic plan for the request of Figure 4 is presented in Figure 6.

We can formally describe a generic plan as a triple:

$$GP = \langle Ip, A, Rp \rangle$$

where Ip is the set of input parameters that should be provided in order to enable the execution of the plan, A is a planed path of activities, and Rp is the set of result parameters that are returned once the plan is executed successfully. The results parameters constitute the answer to the XSRL request. For generic plans, the values of input and result parameters are left unspecified. The sets Ip and Rp need not be disjoint, i.e., it is possible that $Ip \bigcap Rp \neq \emptyset$, as shown in the plan example in Figure 6. The planed path A describes the activities to be performed in order to achieve the goal portion of an XSRL request. It consists of a set of *basic actions* that are combined with *control operators.* The basic actions in the plan correspond to the actions in the activity diagram that are under the control of the user application.

In Figure 6, we show an example of a generic plan for the request of Figure 4. The <input-params> tags enclose the XSRL input parameters where an entire set of input parameters are attached to a binding variable, e.g., the travel data and constraints on the trip are attached to the binding variable $\$a$. Output parameters are specified in the same manner (<result-params>). The action tags <action-path> enclose the action paths generated by the MBP. In the plan of Figure 6, we can distinguish actions that require interaction with the service providers (e.g., `CreatePackage` or `AcceptPackage`) and actions that require an interaction with the user (e.g., `EvaluatePackage`). The control operators permit to specify sets of actions that should be performed in sequence, conditional activities to be performed only under certain circumstances (e.g., actions `ReceiveConfirmationDetails` is executed only if the user accepts a given package), as well as actions that should be iterated until a desired condition is achieved (e.g., the user can keep revising the package, as specified by the external `<repeat>...</repeat>` in the plan). Special constructs `<success>...</success>` and `<failure>...</failure>` represent plan termination with success or failure.

Once the generic plan has been produced, the planner interacts with the service providers and with the user according to the path of actions specified in the generic plan, by appropriately invoking the UDDI enquiry API. The serialized information returned by the service providers via the UDDI is matched against the constraints supplied by the XQuery portion of the XSRL request. Subsequently, the constraint enforcer module checks the consistency of these constraints.

The plan executor, also referred to as constraint enforcer in parts of the paper, produces a number of *instantiated plans* corresponding to the generic plan that satisfy the constraints specified in the XQuery portion of the XSRL. We can formally describe an instantiated plan as a 4-tuple:

$$P = \langle Pid, Ip, A, Rp \rangle$$

where Pid is a unique identifier signifying an instance of a generic plan GP. Ip is the set of instantiated input parameters, A a planed path of actions, and Rp the set of result parameters for the instantiated plan P. The set of input parameters Ip correspond to those of the generic plan, but at this stage values are specified for the input parameters. The planed sequence of actions for the instantiated plan is identical to its corresponding generic plan. The set of return parameters Rp correspond to those of the generic plan with the exception that values are now specified for these parameters.

The instantiated plans are the means of providing answers, viz. values, to a user's request such as, for instance, different flight carrier possibilities, destinations, prices, duration, etc. The process of querying the UDDI and enforcing constraints has produced a number of instantiated plans corresponding to the original generic plan. During plan execution the user has the possibility to interact with the planner and to choose which instantiated plans to accept, which to revise, and which to reject (for example we refer to the activity `EvaluatePackage`). User's choice is expressed by means of the interactive part of XSRL and results in a new

```
<plan>

  <input-params>
    <param name="$ta" type="TravelAgent"></param>
    <param name="$a" type="AirPackage"></param>
    <param name="$ha" type="HotelAgent"></param>                       Ip
    <param name="$h" type="HotelPackage"></param>
    <param name="$cc" type="CreditCard"></param>
  </input-params>

  <action-path>
    <repeat> -- create the travel package
      <do action="CreatePackage">
        <send value="request($a)" to="$ta"></send>
        <result value="air-segment-details($a)"></result>
      </do>
      <ask-user activity="EvaluatePackage">
        <result value="reject">
          <do action="RejectPackage"></do>
          <failure></failure>
        </result>
        <result value="revise">
          <do action="RevisePackage"></do>
        </result>                                                      A
        <result value="accept">
          <do action="AcceptPackage">
            <send value="credit-card-details($cc)" to="$ta"></send>
            <result value="voucher($a)"></result>
          </do>
          <do action="ReceiveConfirmationDetails"></do>
          <repeat> -- create the hotel package
            [...]
          </repeat>
          <success></success>
        </result>
      </ask-user>
    </repeat>

  <result-params>
    <param name="$a" type="AirPackage"></param>                        Rp
    <param name="$h" type="HotelPackage"></param>
  </result-params>

<plan>
```

Figure 6: An example of a generic plan.

interaction with the planner. The BNF definition of the *interactive XSRL* (iXSRL) language is the following:

```
iXSRL   ::=  accept | reject | revise
accept  ::=  '<Accept>' '<PId>' Pid '</PId>' '</Accept>'
reject  ::=  '<Reject>' '<PId>' Pid '</PId>' '</Reject>'
revise  ::=  '<Revise>' '<PId>' Pid '</PId>' 'WITH'
             ( attribute XQuery_expression_operator val )+ (ReturnClause)*
```

where `Pid` is the instantiated plan identifier, a is an attribute or element name present in Ip of the instantiated plan P, and `ReturnClause` is as defined in Section 4.1. The interpretation of the `accept` and `reject` expressions is straightforward. The `revise` statements requires some explanation as it is the only means for a user to request further interaction with the planner. The user may specify alternative values for some of the Ip parameters and may also optionally change the format of the return parameters. For example, one may wish to revise an instantiated plan corresponding to the request of Figure 4 by specifying an alternative residential area as target of the hotel reservation process.

4.3 Implementing the planner framework

An experimental system for execution of service requests and resulting plans is under construction. We have represented the business process specification of the open travel domain (activity diagram in Figure 2) in terms of $NuPDDL$, a non-deterministic extension of the PDDL language which is the standard Planning Domain Description Language for AI applications [12]. $NuPDDL$ describes domains in terms of actions which have preconditions, postconditions, input parameters and effects. These are formal complete descriptions of a domain which can be used to plan actions, verify plans and check environment behaviours.

The $NuPDDL$ domain description and a simulated version of an XSRL request is then fed to the planner. An excerpt of an $NuPDDL$ domain model specification for the OTA schema for the air segment reservation document model is shown if Figure 7:

This example indicates that in the domain model there are three actions to be performed with respect to a user application dealing with air segment reservations, namely `CreatePackage`, `EvaluatePackage` and `AcceptPackage`. The $NuPDDL$ model contains a complete specification of the effects and the possible succession of actions that need to be taken when planning within an e-marketplace domain such as the open travel. This specification contains pre- and post-conditions attached to actions as well as constructs for expressing non-deterministic behaviour (the `oneof` operator in the `EvaluatePackage` action above).

```
(:action CreatePackage
   :parameters (?a - AirPackage ?ta - TravelAgent)
   :effect (done_CreatePackage ?a))

(:action EvaluatePackage
   :parameters (?a - AirPackage ?ta - TravelAgent)
   :precondition (done_CreatePackage ?a)
   :postcondition (and (not (done_CreatePackage ?a))
                       (done_EvaluatePackage ?a)
                       (oneof (= (result_EvaluatePackage ?a) reject)
                              (= (result_EvaluatePackage ?a) revise)
                              (= (result_EvaluatePackage ?a) accept))))

(:action AcceptPackage
   :parameters (?a - AirPackage ?cc - CreditCard ?ta - TravelAgent)
   :precondition (and (done_EvaluatePackage ?a)
                      (= (result_EvaluatePackage ?a) accept))
   :postcondition (and (not (done_EvaluatePackage ?a))
                       (done_AcceptPackage ?a)))
```

Figure 7: An excerpt from a $NuPDDL$ specification for the OTA schema.

The second main component in the implementation is the model based planner (MBP) [11]. The MBP is an advanced prototype system for the generation of plans that either creates a correct plan or returns an explanation of why there is no plan satisfying a given goal. The MBP produces a generic plan according to a business process specification encoded in $NuPDDL$, or rejects the request. Instantiated plans are generated on the basis of a generic plan by simulating hypothetical inputs of service providers from the UDDI repository.

The planning framework involves a domain design and an implementation phase. The design phase is e-marketplace specific and defines the $NuPDDL$ models required for interaction with the planner and execution of e-marketplace specific XSRL requests. The implementation infrastructure is the foundation for the planning framework. Figure 8 illustrates the connection between the design and implementation phases.

During the design phase, formalisms and techniques are used for analysing e-business requirements, processes and models, which are then represented by means of rich description languages such as $NuPDDL$. This is essential input for the implementation infrastructure of the web-services planning framework. Some initial work that falls under this effort can be found in [13].

The implementation phase of the framework that plans actions in response to XSRL requests involves the following components:

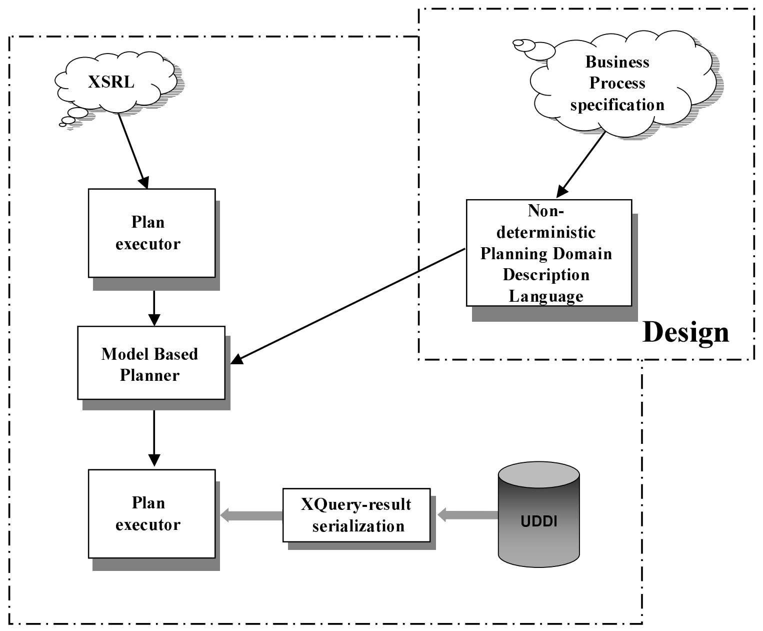
XSRL
Plan executor
Model Based Planner
Plan executor
XQuery-result serialization
UDDI
Business Process specification
Non-deterministic Planning Domain Description Language
Design

Figure 8: Implementation steps.

- **XSRL parser:** the XSRL parser parses XRSL requests and passes them to the planning framework for further processing. It includes a standard parser for XML expressions and a dispatcher module for passing parsed expressions to the MBP and the plan executor.
- **Model Based Planner:** the implementation of the MBP is built on top of a symbolic model checker and relies on Binary Decision Diagrams (BDDs). BDDs provide a general interface that allows for a direct mapping of symbolic representation mechanisms such as prepositional formulae representing sets of states and state transitions. More information on the MBP can be found in [11].
- **Plan executor:** this module is responsible for interacting with UDDI enquiry conformant APIs and coordinates the execution of generic plans obtained from the MBP, which it instantiates and executes. This part of the system is under construction.

5 Related Work

In Section 1 we briefly mentioned work which is related to web-resources planning and scheduling. There are currently no research activities on providing either formal or automated support for service-based request processing other than low-level discovery mechanisms for web-services provided by the UDDI enquiry APIs. It is, however, worth mentioning some loosely related work (mainly on services composition and coordination) in the area of inter-organisational workflows.

Work on flexible workflows has focused on dynamic process modification [14]. In this publication, workflow changes are specified by transformation rules composed of a source schema, a destination schema and of conditions. The workflow system checks for parts of the process that are isomorphic with the source schema and replaces them with the destination schema for all instances for which the conditions are satisfied. They also propose a migration language for managing instance-specific migrations.

The work reported in [15] focuses on methods and tools for defining processes that compose services that are provided by different companies. An advanced workflow model with special purpose primitives manages coordination among services. The model allows definition of application-specific states and operations. Designers specify composite services (processes) by defining control flow conditions based on the states of component services and when application-specific operations should be invoked on the component service.

The approach described in [16] allows for automatic process adaptation. The authors present a workflow model that contains a placeholder activity, which is an abstract activity replaced at run-time with a concrete activity type. This concrete activity must have the same input and output parameter types as those defined as part of the placeholder. In addition, the model allows to specify a selection policy to indicate which activity should be executed.

In [17] the authors describe how workflow technology can be extended in order to support e-business interactions. The authors propose a framework that links B2B interactions with internal workflow systems. The proposed framework can be used for the development of new business processes that potentially support B2B interaction standards, e.g., RosettaNet PIPs, CBL and cXML, and the enhancement of existing business processes by seeding B2B interaction capabilities.

Work related to coordination/composability can also be found in CSCW and groupware publications [18]. In this publication, the authors examine the potential of using coordination technology to model electronic business activities and illustrate the benefits of such an approach. Furthermore, the authors demonstrate that control-oriented, event-driven coordination models and languages are more suitable for supporting electronic business applications rather than the conventional data-driven approaches which are based on accessing an open shared communication medium in almost unrestricted ways.

6 Summary and future work

In this paper, we have presented XSRL, a formal language for expressing request against e-marketplace registered web-services. The language is an amalgamation of internet XML query language and AI planning constructs. We have given full semantics of the constructs of the XSRL that subsume the planning composition language we proposed in [6]. There we introduced constructs at the systems-level to support web-service composition, whereas the XSRL includes constructs such as alternative activities, vital vs. optional activities, preconditions, postconditions, invariants, and expression operators over quantitative values that can be employed at the user-level when formulating a goal. The novelty of this approach lies in the automatic generation and verification of plans over web-services residing in an e-marketplace once a user request is concretely expressed and formally specified. This framework provides genericity and flexibility and can be used as a basis for effective planning for different types of applications dealing with interacting web-services.

Currently, XSRL distinguishes between vital and optional activities and allows user preferences to be specified in a strictly sequential order (total order). However, one may consider further extensions to the language where partial orders of preferences can also be specified. For instance, one may wish to specify that a traveller has an equal preference over Hilton and Marriott chains and that these are preferable to Best Western hotels. Future extensions of XSRL that can be of importance for interacting with web-services may also include similarity operators. There are three levels at which such semantic operators may work:

1. at the level of XQuery functions: a user could specify that s/he wants something similar to a compact car (topology), or some location close to a given city (proximity), etc.

2. at the service level: a user could specify that s/he wants a semantic functionality similar to that provided by a specific service, e.g., a train service may replace a plane service for the goal of travelling—provided that there is a semantic similarity of the operations of the services.

3. at the plan level: a user could specify a goal similar to that obtained by an already

existing (and successful) plan, e.g., make a trip similar to the one s/he has previously requested or done.

References

[1] S. Smith, D. Hildum, and D.R. Crimm. Toward the design of web-based planning and scheduling services. In *Int. Workshop on Automated Planning and Scheduling Technologies in New Methods of Electronic, Mobile and Collaborative Work*, 2001.

[2] D. McDermott. Estimated-regression planning for interactions with Web Services. In *6th Int. Conf. on AI Planning and Scheduling*. AAAI Press, 2002.

[3] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. G. Philpot, and S. Tejada. The ariadne approach to web-based information integration. *International the Journal on Cooperative Information Systems*, 2002. Special Issue on Intelligent Information Agents: Theory and Applications, Forthcoming.

[4] C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *Data Engineering Bulletin*, 2002. To appear.

[5] 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.

[6] J. Yang and M. Papazoglou. Web component: A substrate for web service reuse and composition. In *14th Int. Conf. on Advanced Information Systems Engineering CAiSE02*, 2002.

[7] Aiello et al. A request language for web-services based on planning and constraint satisfaction. In *VLDB Workshop on Technologies for E-Services (TES02)*, 2002.

[8] OTA Open Travel Alliance. 2001C specifications, 2001. `http://www.opentravel.org`.

[9] XQuery. An XML Query Language, 2002. `http://www.w3.org/TR/xquery`.

[10] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier, 1990.

[11] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: A Model Based Planner. In *n Proc. IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[12] 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, *4^{th} Int. Conf. on AI Planning and Scheduling*, 1998.

[13] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specifications in Tropos. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering (RE01)*, 2001.

[14] G. Joeris and O. Herzog. Managing evolving workflow specifications with schema versioning and migration rules, 1999. TZI Technical Report 15, University of Bremen.

[15] D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. Integrating Workflow Management Systems with Business-to-Business Interaction Standards. *Information Systems*, 24(6), 1999.

[16] D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. Managing escalation of collaboration processes in crisis mitigation situations. In *Proceedings of Int'l Conf. On Data Engineering ICDE 2000*, 2000.

[17] M. Sayal, F. Casati, U. Dayal, and M Chien Shan. Integrating Workflow Management Systems with Business-to-Business Interaction Standards. In *Proceedings of Int'l Conf. On Data Engineering ICDE 2000*, 2002.

[18] G. A. Papadopoulos and F. Arbab. Modelling electronic commerce activities using control-driven coordination. In *Ninth International IEEE Workshop on Database and Expert Systems Applications*, 1998.