SEAMLESS INTEGRATION OF INQUIRY AND TRANSACTIONAL TASKS IN WEB APPLICATIONS*

Juan J. Rodriguez, Oscar Diaz
The EKIN team
Dpto. de Lenguajes y Sistemas Informaticos
University of the Basque Country
Apdo. 649 - 20080 San Sebastin (Spain)
jibrojij@si.ehu.es, jipdigao@si.ehu.es

Abstract Most conceptual Web design methods proposed so far focus on browsing (i.e. inquiry tasks) but it is not clear how to integrate them with transactional tasks which have a lasting effect. Moreover, tasks are commonly integrated into higher-order behavioural units: the processes. For instance, the process of a purchase includes "browsing the catalog", "adding to the trolley", "filling up billing data" and other tasks that end up in the fulfillment of the order. We claim that these distinct task types (i.e. inquiry and transactional tasks) impose different demands and require distinct skills from the designer. On these grounds, we envisage a bottom-up approach to web application construction. First, inquiry and transactional task design is conducted by two separate teams each with expertise in one area. Second, processes are realised through inter-task dependencies. Declarative and separate description of tasks and dependencies accounts for maintenability of the whole solution. This paper presents how this approach has been realised in *AtariX*, a tool environment for the specification and support of web applications. Transactional tasks reside in the middle-tier implemented as Enterprise JavaBeans whereas both inquiry tasks and inter-tasks dependencies are regulated at the Web server.

*This research was partially supported by the Secretaría de Estado de Política Científica y Tecnológica of the Spanish Goverment under contract TIC 1999-1048-C02-02. Juan J. Rodríguez enjoys a pre-doctoral grant by the Basque Goverment.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35658-7_21
R. Meersman et al. (eds.), *Semantic Issues in E-Commerce Systems*

1. Introduction

An organization's e-commerce service could be a mix of plain information access, online consultation with the organization's representatives, and access to complex backend operations normally supported by an ERP[10][11]. As an interactive means, a web site, offers support for three distinct types of interaction: inquiry-based, transactional and consultive. An inquiry interaction aims to recover some data for informative purposes only. E-catalogs and e-brochures but also support for OLAP applications fall within this realm. On the other hand, transactional interactions frequently involve committing database state changes. An employee entering orders or a customer making an order are examples of transactional interactions. Finally, consultive interactions involve groupware such as the joined preparation of documents or support for agreement negotiations. This paper focuses on inquiry and transactional interactions.

Most conceptual Web design methods proposed so far do not address explicitly the integration of interactions of distinct nature (i.e. inquiry-based, transactional and consultive). Work described in [7],[5] or [12] come from the hypertext area. Their main focus is on inquiry interactions by providing powerful built-in navigation primitives. For data-intensive web sites (i.e. sites displaying mainly database data) the WebML system stands out [3]. Similarly to our approach, WebML is model-driven and it also contemplates the integration of transactional interactions in the navigation space [2]. However, these transactions are conceived as independent, isolated units where the role of the Web site is restricted to be a front-end for invoking operations. By contrast, we promote the view of the web site as a task integration space. For this purpose, we propose the concept of "*workview*" as a basic construct for the declarative specification of the restrictions and dataflows that tie up a set of interrelated tasks. The task itself should be unaware of those dependencies. Briefly stated, a workview is a set of tasks plus a set of dependencies. Both tasks and dependencies can be easily added or removed and in so doing, changing the site behaviour. This allows for maintainable and evolvable web sites, which have an increasing demand in the competitive e-commerce field. This idea of workview resembles the notion of activity found in object-oriented database systems [8] as well as in business process modeling [6].

Therefore, we promote a bottom-up approach to web application construction: (1) task (either inquiry or transactional) construction, (2) task integration. In so doing, we aim at enhancing the cohesiveness of the user experience.

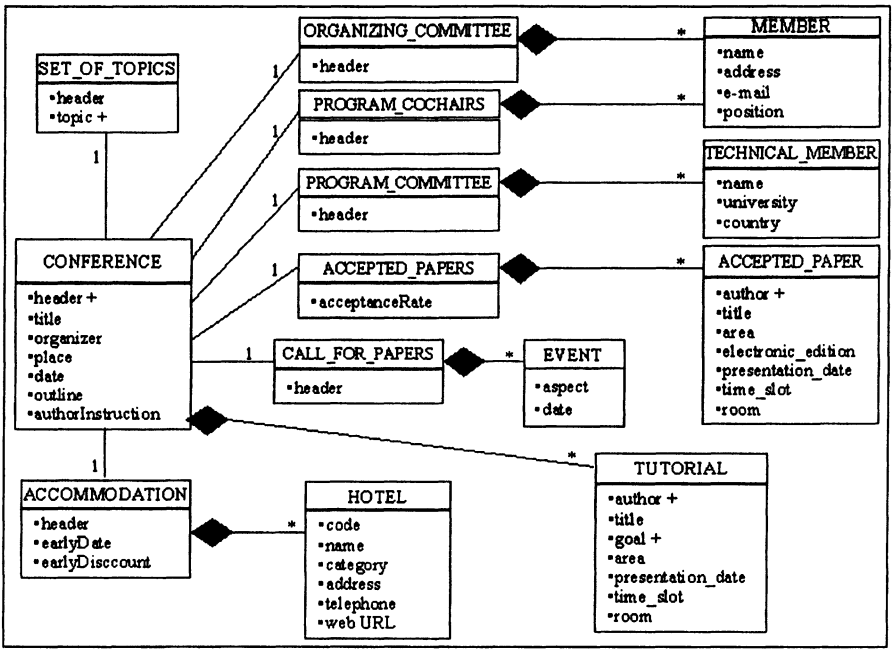

Figure 1. The content diagram for the conference example.

These ideas have been borne out by *AtariX*, a tool environment for the specification and support of web applications. The IFIP conference web site is used as a running example throughout the article. The result can be found at *http://sipl68.si.ehu.es/atarix/ifipExample.*

The rest of the paper is structured as follows. Section 2 is concerned with the notion of task. Section 3 discusses task integration and presents the notion of workview. The paper concludes with section 4 containing final remarks and a discussion about future work.

2. Task construction

For the purpose of this paper, a task is defined as the resalisation of a purposeful unit of interaction with the user. What makes an interaction purposeful? Here we are faced with the issue of interaction granularity: is providing the user name a "purposeful unit of interaction"? or is it any keystroke? To correctly identify meaningful units of interaction without going too deep into the refinement process, a task aim can be ascertained by the following question: is it sensible for a user to leave the application at this point? Whereas leaving the application after *"entering your name"* does not appear very meaningful, the user can have a break once

a "customer has been registered". Of course, registering a new customer will imply distinct interactions, too, for example, to collect data from the user, but every low-level interaction is aimed at achieving a common goal: the user registration. Therefore, "purposeful units of interaction" are ascertained from the goals of the stakeholders in using the web site.

According to its associated goal, a task can be classified as:

- an inquiry task; an interaction unit for inquiry purposes only. Browsing through the activities of a conference can be regarded as an inquiry task whose aim is for the user to become aware of the conference's events. These tasks are supported through HTML page navigation.

- a transactional task; an interaction unit which can lead to a change in the state of the domain. Both registering for a conference or hotel booking, are examples of transactional tasks which end up having an effect on the storage backend. Transactional tasks are commonly within the realm of ERP packages and lie in the application server layer. In our implementation, these tasks are realised as component services, more specifically, as EJB services.

- a session task; an interaction unit whose change lifespan is restricted to the current session. Adding a product to the trolley is the most common example of a session task. Its effects are restricted to the session without any persistence implications. Hence, we decided to support these tasks as JavaBeans where the session state is kept.

The following subsections look at each task type in detail.

2.1. Inquiry task definition

The purpose of inquiry tasks is to make the user aware of some data. Rendering a new chunk of data is seen as an *inquiry task*. The system realises that an inquiry task occurrence happens by means of navigation. That is, inquiry task occurrences arise when the user browses the content document (see section 3.1). Therefore, the inquiry task definition has (1) to establish "the content space", and (2) has to specify the potential paths along which the navigation can proceed.

In *AtariX*, "the content space" is realised through an XML document. Figure 1 shows the structure of the content for the conference example. The *CONFERENCE* node represents the root element of the XML document which has as sub-elements *SET_OF_TOPICS*, *ACCEPTED_PAPERS* and so on. *ACCEPTED_PAPERS* in turn consists

of a set of *ACCEPTED_PAPER* sub-elements. This diagram is realised by a content document[1]. The elements can be defined locally or externally. For instance, the *ACCEPTED_PAPERS* elements are defined by the following SQL query: *select * from PAPERS where status='accepted'*. At execution time this query returns a set of ACCEPTED_PAPER elements.

The second aspect addresses path definition. Following the approach described in hypermedia applications, a *link* construct is introduced. In *AtariX*, the link's origin and destination are elements of the XML content document. Elements within the XML content document are addressed using the W3C standard XPath notation [13]. For instance, the link that traverses from the CONFERENCE element to the CALL_FOR_PAPERS element is described as follows:

```
<LINK title="Call For Papers" from="/CONFERENCE" to="CALL_FOR_PAPERS">
        <CONTENT order="1" couplingMode="embedOneOnRequest"/>
</LINK>
```

LINK is an element of the *AtariX* vocabulary. This element has a set of attributes which describe (1) the label of the link when rendered on the screen (the *title* attribute); the origin of the link (the *from* attribute) denoted by an XPath expression that indicates when the link is available (in this case the link is available when the *CONFERENCE* element is rendered); and the destination of the link (the *to* attribute) which states the element to be rendered when this path is followed. Additional aspects of the AtariX navigation specification can be found in [4].

Therefore, the content and navigation document implicitly define the potential set of inquiry tasks. The rendering of any chunk of content (e.g. displaying data about a tutorial) as a result of path traversal is seen as a potential occurrence of an inquiry task. Hence, every content element referenced by the navigation specification is a potential inquiry task. The decision of what inquiry tasks are truly relevant for the workflow, is postponed until the workview design time (see section 3.1).

2.2. Transactional task definition

The purpose of a transactional task is to achieve some meaningful domain state change. Transactional tasks resemble database transactions. A database transaction is an atomic set of database changes. As such, it can affect distinct objects in the domain. The essence of transactions

[1]About XML data model limitations or other issues concerning the content model (e.g. heterogeneous data sources or data obsolescence) see [4].

```
<?atarix_component_document href="ifipComponents.xml" type="text/xml"?>
<TASKS>
  <TASK ID="accomodationBookingTra">
    <TASK_PARAMETER ID="registrationNumber" type="Long"/>
(a) <TASK_PARAMETER ID="hotelCode" type="String"/>
    <TASK_PARAMETER ID="roomType" type="String"/>

    <ALTERNATIVES>
      <ALTERNATIVE ID="accepted">
        <CONTEXT>
  (b)     <!--Registration is OK and Room is Available -->
        </CONTEXT>
        <ACTIONS>
          <ACTION ID="book" component="BookingManagement" service="bookingRoom">
            <PARAMETER name="code">
              <FROM_PARAMETER parameterIDREF="hotelCode"/>
  (c)       </PARAMETERr>
            <PARAMETER name="roomType">
              <FROM_PARAMETER parameterIDREF="roomType"/>
            </PARAMETER>
          </ACTION>
        </ACTIONS>
        <RESULTS>
          <TASK_RESULT name="BokingNumber">
            <FROM_SERVICE_RESULT actionIDREF="book" serviceResult="bookingNumber"/>
  (d)     </TASK_RESULT>
          <TASK_RESULT name="message">
            <MESSAGE>Booking performed successfuly</MESSAGE>
          </TASK_RESULT>
        </RESULTS>
      </ALTERNATIVE>
      <ALTERNATIVE ID="rejected">
        <CONTEXT>
          <!-- Registration is not OK or No Room Available -->
        </CONTEXT>
      ...
    </ALTERNATIVE>
    </ALTERNATIVES>
  </TASK>
...
</TASKS>
```

Figure 2. The transactional task document for the conference example.

is to guarantee atomicity, i.e. no partial execution of the transaction is allowed.

However, a database transaction is an implementation mechanism to support transactional tasks. Quite often, this is realised as a potentially disjoint set of database transactions. For example consider the *borrowBookRequest*. This task encompasses all the interactions involved in handling the request of borrowing a book. Its fulfillment does not necessarily end up in lending the book to the student (which would be realised by the invocation of the *borrowBook* database transaction). If the book is not currently available the following alternatives are possible: (1) handling the request, (2) rejecting the request, (3) buying the book or (4) opting for an inter-library loan.

Figure 2 shows the definition of the *accomodationBookingTra* task in *AtariX*. This task supports hotel booking for the conference attendees. The definition of a transactional task includes: a selector (e.g. *accomodationBookingTra*), a set of parameters (e.g the attendee registration number, the hotel identification and the type of room requested by the attendee) (see figure 2a), and a set of the different execution alternatives for the task. In this case, the fulfillment of the task results either, in the successful booking of the accommodation, or in the rejection of the booking request due to a shortage of rooms. Each alternative is described by tree tags: *CONTEXT, ACTIONS* and *RESULTS*.

CONTEXT (see figure 2b) holds the conditions that lead to the execution of this alternative. *ACTIONS* contains the services to be invoked when the alternative is taken (in our example, the invocation of the *bookingRoom* service on the *BookingManagement* component, see figure 2c). Each of these services is supported by an Enterprise JavaBean component [9]. Finally, *RESULTS* holds the outcomes of the alternative execution. The results from the different actions are collected and summarized in this section. In our example, it returns a booking identifier as its main result (see figure 2d).

2.3. Session task definition

The purpose of a session task is to achieve some session state change. Its definition is similar to the transactional task definition. Its only difference is that it is supported through client-side components (such as JavaBeans). Adding to the trolley some of the conference "products" (i.e. the conference itself, a tutorial or a workshop). Then, *"trolleyAddingSes"* is a session task. The attendee can register for distinct events and then perform a single *"conferenceRegTra"* task.

3. Task integration: the notion of workview

Previous section introduced different types of tasks. This section addresses the integration of these tasks.

If the functionality supported by a web site were reduced to a single goal, the notion of task would be sufficient. However, this is rarely the case as most sites allow the user to accomplish a set of tasks rather than just a single task. Thus, a site comprises a set of services that potentially proceed from different sources. However, the site appears to function as a single whole via the web site that represents the workview.

The definition of the workview has to do with at least three issues: (1) identifying the tasks to be integrated, (2) specifying control dependencies and, (3) defining navigation dependencies. Task identification

addresses how to select a cohesive group of tasks that provide the appropriate workview. The other two issues are concerned with how tasks are harnessed into a single workview by specifying and enforcing inter-task dependencies to manage the flow of data and control between the tasks. In back-end systems (such as workflow management systems) this aspect involves specifying and enforcing inter-task dependencies to manage the flow of data and control between the tasks (we refer to them as control dependencies). However, in a front-end context such as the Web, inter-task dependencies not only include control dependencies but also navigation dependencies. These dependencies state how tasks are "anchored" in the hypermedia space defined by the navigation space. Navigation dependencies address where tasks can be invoked or where to return to if the task fails.

3.1. Establishing the boundaries of the "workview"

During this stage the designer has to identify which are the tasks to be accomplished through the web application. The criterium to be used can be object-centric, process-centric or role-centric. It depens on whether the included tasks correspond to those affecting a given type of object, fulfill the set of steps conforming a given process, or are issued by users belonging to a certain role. Of course, a Web designer does not need to stick to a single criterium. Most Web sites use different criteria for distinct parts of the site.

For our conference example, the following tasks are included:

- inquiry tasks: rendering the conference outline (*"outlineInq"*), rendering conference tutorials (*"tutorialsInq"*), rendering the associated workshops (*"workshopInq"*) or rendering available accommodation (*"hotelInq"*) illustrate this task type. It should be pointed out that at this stage, the designer focus on ascertaining which inquiry tasks are required (i.e. the information needs) but no how these tasks are realised. This aspect is postponed until the definition of the navigation dependencies (see section 3.3).

- transactional tasks: registering for the conference *("conferenceRegTra")*, booking accommodation (*"accomodationBookingTra"*) or registering for a push service that keeps the user updated with the latest news via e-mail (*"pushServiceRegTra"*), are examples of this task type.

- session tasks: a common example is adding to the trolley some of the conference "products" (*"trolleyAddingSes").*

```
<?atarix_task_document href="ifipTasks.xml" type="text/xml"?>
<WORKVIEW ID="conference">
 <TASK taskID="tutorialsInq" lifespan="intra-session"/>
 <TASK taskID="outlineInq" lifespan="intra-session"/>
 <TASK taskID="hotelInq" lifespan="intra-session"/>
 <TASK taskID="trolleyAddingSes" lifespan="intra-session"/>
 <TASK taskID="conferenceRegTra" lifespan="inter-session"/>
 <TASK taskID="accommodationBookingTra" lifespan="inter-session"/>
 <TASK taskID="pushServiceRegTra" lifespan="inter-session"/>
 <CONTROL_DEPENDENCIES>
   ...
   <CONSTRAINT taskIDREF="hotelInq">
     <ENABLED_WHEN>
       <EXISTS task="conferenceRegTra"/>
     </ENABLED_WHEN>
     <HISTORY_MANAGEMENT>
       <IN task="conferenceRegTra"/>
       <OUT on="conferenceRegTra">
         <ACTION type="clear_most_recent" task="conferenceRegTra"/>
       </OUT>
     </HISTORY_MANAGEMENT>
   </CONSTRAINT>
   <CONSTRAINT taskIDREF="accommodationBookingTra">
     <ENABLED_WHEN>
       <AND>
         <EXISTS task="conferenceRegTra">
           <OCCURRENCE_PREDICATE>
(a)          <!-- Where alternative property equal to successfull -->
           </OCCURRENCE_PREDICATE>
         </EXISTS>
         <EXISTS task="hotelInq"/>
       </AND>
     </ENABLED_WHEN>
     <DATAFLOW>
       <PARAMETER name="registrationNumber" fixed="true">
         <FROM_OCCURRENCE task="conferenceRegTra" occurrence="newest">
           <FROM_PROPERTY name="regNumber"/>
         </FROM_OCCURRENCE>
(b)    </PARAMETER>
       <PARAMETER name="hotelCode" fixed="true">
         <FROM_OCCURRENCE task="hotelInq" occurrence="newest">
           <FROM_PROPERTY name="hotelCode"/>
         </FROM_OCCURRENCE>
       </PARAMETER>
     </DATAFLOW>
     <HISTORY_MANAGEMENT>
       <IN task="conferenceRegTra"/>
       <IN task="hotelInq"/>
(c)    <IN task="accommodationBookingTra"/>
       <OUT on="hotelInq">
         <ACTION type="clear_most_recent" task="hotelInq"/>
       </OUT>
     </HISTORY_MANAGEMENT>
   </CONSTRAINT>
 </CONTROL_DEPENDENCIES>
 ...
</WORKVIEW>
```

Figure 3. The workview document for the conference example (part 1).

3.2. Specifying control dependencies

Control dependencies determine how tasks are interwoven. The description of the execution can be either procedural or declarative. In the first case, constructs like those found in programming languages are used to specify execution control. By contrast, a declarative description constrains the space of possible interactions through both temporal and existence conditions [1]. The algorithmic-like approach fits a process-centric site but it is counter-intuitive for object-centric as well as role-centric sites. On the other hand, declarative descriptions of execution control accounts for flexibility and maintainability at the price of complex debugging. Whereas an algorithm-like description provides a clear picture of the event flow at compile time, this is not the case for constraint-based descriptions. Usually, a global scheduler is responsible for the correct execution of the workflows according to all dependencies stated.

AtariX follows a declarative approach to inter-task dependency specification. We think this approach naturally fits the event-driven nature that characterises GUI applications. A dependency reflects a necessary condition for a task to be invoked. Our approach is to describe those conditions as predicates on the flow of task occurrences which have happened during the session. This flow of ordered task occurrences is known as the *history*. In this way, the designer can state that the *accommodationBookingTra* task can only be selected if a *hotelInq* is already kept on the history (i.e. it has been previously executed). Notice that this does not imply that *accommodationBookingTra* must follow *hotelInq* but that the execution of *hotelInq* is a necessary condition for the invocation of *accommodationBookingTra*. However, the task itself should be unaware of those dependencies. For this purpose, we propose the concept of workview as a basic construct for declaratively specifying the restrictions and dataflows that tie up a set of inter-related tasks. Briefly stated, a workview is a set of tasks plus a set of dependencies.

Figure 3 shows the *workview* document for the conference example. All the involved tasks are identified trough the TASK tag whereas the set of dependencies that tie these tasks together are reflected by the CONSTRAINT tag. A constraint is in turn described by a triple *<enabledWhen, dataFlow, historyManagement>* .

*The **"enabledWhen"*** tag holds a condition on the history. A history repository is kept locally for each task. This repository records those task occurrences of interest for enabling/disabling the hosted task. When a task occurrence is generated, the system stores a copy in each repository associated with those tasks that can be potentially affected by this

occurrence. Each task occurrence contains information about the input parameters, the result and the alternative chosen during the execution of the task. All these aspects can be checked by the condition.

For example, on the conference web site the following three tasks can be issued: *conferenceRegTra, hotelInq, accommodationBookingTra.* We can think of a situation where booking the accommodation should be preceded by both conference registration and hotel inquiring. This can be described as:

$$\exists conferenceRegTra(alternative = \text{"successful"})\ and\ \exists hotelInq$$

which states that before *accommodationBookingTra* is available, the history should contain both a successful *conferenceRegTra* and *a hotelInq* task occurrences (the description in XML is shown in figure 3a).

The ***"dataFlow"*** tag indicates the possible data flow between the tasks which compose the workview. A task parameter can be obtained by directly querying the user. Other alternative is to take it from parameters of previous task occurrences kept in the history. As an example, consider the *conferenceRegTra* task. It has three parameters: the *registrationNumber,* the *hotelCode* and the *roomType.* Instead of directly prompting the user, the *registrationNumber* and the *hotelCode* can be obtained from previous occurrences of *conferenceRegTra* and *hotelInq,* respectively. As for *roomType,* this is directly provided by the user. The specification of this data flow can be found in 3b.

The ***"historyManagement"*** tag. Specifying how data flows between tasks is not enough. As a motivating example, consider that *accommodationBookingTra* takes its *hotelCode* parameter from the *hotelInq* task. The user issues distinct hotel inquiries before choosing a particular one: first *hotelInq(hotel1)* , then *hotelInq(hotel2),* and finally, *hotelInq(hotel3).* At this time, the *accommodationBookingTra*'s history could have the following entries: *{hotelInq(hotel1), hotelInq(hotel2), hotelInq(hotel3)}* where *hotel1*, *hotel2* and *hotel3* are the codes of the different hotels that have been consulted. If now the *accommodationBookingTra* task is issued, which of the available *hotelInq* occurrences should be used to obtain the *hotelCode* parameter? If next, the user selects the *accommodationBookingTra* again, should the system use the same *hotelCode*? These questions pose the need for a **selection policy** and a **consumption policy**. The former tackles the situation where several task occurrences of the same type are in the history (as in our previous example with *hotelInq*). In this case, which is the task occurrence to be used to extract the parameter? Two selection policies are defined to prevent any ambiguities: *"newest"* which selects the lastest occurrence, and *"oldest"* which takes the occurrence that appears first in the history. In our example, defining a *"newest"* selection policy would instruct the system to

book the room at the *hotel3* whereas the *"oldest"* alternative would select *hotel1*.

The consumption mode indicates whether the task occurrence "consumed" (e.g. *hotelInq(hotel3)*) when invoking a task (e.g. *accommodationBookingTra*) should be removed from the history log or not. For instance, *accommodationBookingTra* takes the value of the parameters *registrationNumber* and *hotelCode* from previous occurrences of *conferenceRegTra* and *hotelInq*, respectively. Whereas the *conferenceRegTra* occurrence is never removed from the history (the registration number is obtained once and again from the very same *conferenceRegTra* occurrence, regardless on how many *accommodationBookingTra* are issued), *hotelInq* occurrences are removed from the local history once consumed (i.e. every *accommodationBookingTra* takes the hotel from a different *hotelInq* occurrence). Once all the *hotelInq* occurrences have been consumed, the *accommodationBookingTra* is disabled. In figure 3c, the tags *"in"* and *"out"* specify the history management policy. Since this behaviour (i.e. what should be kept in the history and the management of the history) depends on the task type, the history is defined locally for each type of task.

A final issue is *the task occurrence lifespan* which addresses whether a task occurrence survives among different sessions. For example, an attendee that had registered for the conference at the first session connects again to the conference site to find accommodation. Of course, she should not be forced to register again. The former registration occurrence should be kept within the workview history so that the tasks *hotelInq* and *accomodationBookingTra* are readily available. Hence, the lifespan of the registration occurrence should be *"inter-session"*. By contrast, inquiry tasks are more likely to have an *"intra-session"* lifespan. You could be interested in forcing the user to visit again some data before undertaking a transactional task. Finally, session tasks by their very nature have an *"intra-session"* lifespan. Notice, that the lifespan of a task can vary among the workviews the task participates in. Hence, the lifespan is specified in the workview as an attribute of the participating tasks.

3.3. Specifying navigation dependencies

Navigation dependencies state the binding between browsing and task enactment. This binding implies different things for distinct task types.

For an inquiry task, this binding determines the chunk of content whose browsing causes the occurrence of the task. Notice that, it is at this very moment when content rendering becomes an inquiry task. As an example, consider that a task ocurrence should be generated when browsing

```
<WORKVIEW>
  ...
  <NAVIGATION_DEPENDENCIES>
    ...
    <TRANSACTION_ANCHOR taskID="accommodationBookingTra">
      <ENTRY_NODE node="/CONFERENCE/ACCOMMODATION/HOTEL" title="registration"/>
      <CANCEL_NODE node="$invocationContext" title="cancel"/>
      <EXIT_NODE node="/CONFERENCE" title="toConference" taskResultID="booking"/>
      <EXIT_NODE node="$invocationcontext" title="toHotel" taskResultID="rejected"/>
(a)  </TRANSACTION_ANCHOR>

    <SESSION_ANCHOR taskID="trolleyAddingSes">
      <ENTRY_NODE node="/CONFERENCE/TUTORIAL" title="addToTrolley"/>
      <CANCEL_NODE node="$invocationContext" title="cancel"/>
      <EXIT_NODE node="$invocationContext" title="toConference" taskResultID="added"/>
    </SESSION_ANCHOR>

    <INQUIRY_ANCHOR taskID="hotelInq">
      <INQUIRED_NODE node="/CONFERENCE/ACCOMODATION/HOTEL"/>
(b)   <TASK_PARAMETER name="hotelCategory" source="$invocationContext/@category"/>
      <TASK_PARAMETER name="hotelCode" source="$invocationContext/@code"/>
    </INQUIRING_ANCHOR>
    ...
  </NAVIGATION_DEPENDENCIES>
</WORKVIEW>
```

Figure 4. The workview document for the conference example (part 2).

HOTEL information (i.e. the *hotelInq* task). Figure 4b indicates how this is specified for the *hotelInq* task. The *INQUIRED_NODE* tag holds an XPath expression (e.g. *CONFERENCE/ACCOMMODATION/HOTEL*) which locates the element of the content document whose rendering causes the *hotelInq* occurrence. The occurrence parameters are obtained from the *HOTEL* node instance at run-time.

As for transactional and session tasks, navigation dependencies indicate the *ENTRY_NODE*, the *CANCEL_NODE* and distinct *EXIT_NODE*s nodes from where a task can be invoked, cancelled or exited, respectively. For example, the *accommodationBookingTra* (see figure 4a) task can be invoked when hotel information is being displayed (the *ENTRY_NODE* tag). Afterwards, the system prompts for the task parameters. If cancelled, the browser goes back to the entry node (kept in the system variable *$invocationContext*) as specified by the *CANCEL_NODE* tag. Finally, the *EXIT_NODE* depends on the task alternative obtained at run-time. In this case, two alternatives are possible. Either the booking is successfully made (i.e. taskResultID="booking") in which case the browser returns to the *CONFERENCE* node, or the booking is rejected (i.e taskResultID="rejected") which causes the browser to return to the entry node.

Summing it up, a workview supports a cohesive view of the set of tasks which can be accomplished through a web site. "Cohesiveness" is

achieved by means of control and navigation dependencies. The specification of these inter-task dependencies follows a declarative approach.

4. Conclusions

This work presents how both inquiry and transactional tasks have been integrated in *AtariX*, a tool environment for the specification and support of web applications, *AtariX* currently runs on Explorer 5.0 or higher with the Microsoft XML parser 3.0. The following aspects are regarded as the main contributions of this work:

- the notion of *workview*. *AtariX* promotes the view of a web site as a task integration space. To this end, a *workview* is a main construct to integrate a set of tasks into a single whole.
- a history-based description for the specification of control-flow dependencies. This no-prescriptive approach is akin to the event-driven nature of the browser and accounts for a loose coupling among tasks.
- the notion of navigation dependencies. This aspect tackles how tasks are anchored in the hypermedia space. These dependencies can have an important impact in the usability of the system.

Our future plans include: (1) making AtariX available for other platforms, (2) enhancing the expressiveness of the dependency vocabulary to fit the requirements imposed by e-commerce applications.

References

[1] P. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Conf. on Very Large Data Bases (VLDB)*, pages 134–145, 1993.

[2] A. Bongio, S. Ceri, P. Fraternali, and A. Maurino. Modeling data entry and operations in WebML. In *WebDB (Informal Procedings)*, pages 87–92, 2000.

[3] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33(1-6):137–157, 2000.

[4] O. Diaz, F. Ibañez, and J. Iturrioz. A model-based approach to portal development. In *This Volume (9th IFIP 2.6 Working Conference on Database Semantics (DS-9))*, 2001.

[5] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Overview of strudel: A web-site management system. *Networking and Information Systems Journal*, 1(1):115–140, 1998.

[6] P. Hartel and R. Jungclaus. Modeling business processes over object. *International Journal of Cooperative Information System*, 4(2):165–188, 1995.

[7] T. Isakowitz, E.A. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–43, 1995.

[8] L. Liu and R. Meersman. The building blocks for specifying communication behavior of complex objects: An activity-driven approach. *ACM Transactions on Database Systems*, 21(2):157–207, June 1996.

[9] Sun Microsystems. Enterprise JavaBeans Technology. http://java.sun.com/products/ejb/index.html.

[10] W. Rajput. *E-Commerce Systems Architecture and Applications.* Artech House Publishers, 2000.

[11] R. Kalakota M. Robinson. *e-Business: Roadmap for Success.* Addison-Wesley, 1999.

[12] G. Rossi, D. Schwabe, and F. Lyardet. Web application models are more than conceptual models. In P.P. Chen, D.W. Embley, and S.W. Liddle, editors, *World Wide Web and Conceptual Modeling*, pages 193–208, October 1999.

[13] W3c. XML Path Language (XPath) Version 1.0 at http://www.w3.org/tr/xpath.html, 1999.