Exception Handling in Workflow-Driven Web Applications

Marco Brambilla Stefano Ceri Sara Comai Christina Tziviskou
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio 34/5,
20133 Milano, Italy
+39 02 2399 3408 | 3532 | 3474 | 3649

{mbrambil, ceri, comai, tzivisko}@elet.polimi.it

ABSTRACT

As the Web becomes a platform for implementing B2B applications, the need arises of Web conceptual models for describing Web oriented workflow applications implementing business processes. In this context, new problems about process correctness arise, due to the loose control of Web applications upon the behavior of their Web clients. Indeed, incoherent user's behavior can lead to inconsistent processes.
This paper presents a high level approach to the management of exceptions that occur during the execution of processes on the Web. We present a classification of exceptions that can be raised inside workflow-driven Web applications, and recovery policies to retrieve coherent status and data after an exception. We devise these concepts at high level and then we exploit them using a Web modeling language (WebML) that in turn provides development facilities like automatic code generation, validation of hypertext models, and so on. An industrial implementation experience is briefly presented too.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – computer-aided software engineering (CASE), evolutionary prototyping, object-oriented design methods, user interfaces.

D.2.5 [Software Engineering]: Testing and Debugging – error handling and recovery, tracing.

H.4.0 [Information Systems Applications]: General.

H.5.4 [Information Interfaces and Presentation]: Hypertext / Hypermedia – architectures, navigation, theory, user issues.

General Terms: Management, Design, Reliability.

Keywords: Workflow, Exceptions, Failure, Web applications, Navigation behavior.

1. INTRODUCTION

In recent years, the Web is more and more being used as the implementation platform for B2B applications, aiming at supporting business processes, content management, document approval flows, value-added services, and so on. This leads to the integration of several technologies, which go far beyond the simple implementation of Web interfaces for content publishing to the users. Therefore, design, implementation, and maintenance of Web applications are becoming more and more complex. Several different expertises are needed for coping with these new requirements and technologies.

In this context, Web applications assume a mission-critical role within the enterprise and they cannot be considered as mere content-browsing interfaces any more. Therefore, the need arises of solid approaches to users' behavior modeling, to fault management and to exception handling.

Contributions from software engineering and other fields can partially address these issues, although we claim that the Web context raises new and original problems, which require some innovation to the "traditional" approaches and methodologies. Indeed, conceptual modeling expertise from other fields has been widely recognized as valid starting point for defining conceptual aids for Web application development, but the first generation of conceptual models for the Web [1][2][4][6][7][8][10][11][14] essentially focuses on capturing the structure of data to be published and the navigation primitives, represented by such concepts as pages, content nodes, links, and operations.

To cover business processes support, a second generation of conceptual models is required. Such second generation copes with process and workflow modeling, supports Web service interaction, manages exceptions and recovery policies, and integrates data-centric and process-centric modeling primitives into a mix suited to the development of advanced B2B Web applications.

Exceptions that can happen in a Web-based application have peculiar characteristics with respect to traditional workflow applications. This is due to three main aspects: (i) interaction options provided by browser-based interfaces are very powerful, but they are more oriented to free navigation than to strict processes adherence (e.g., users are enabled to jump back and forth on navigated pages, thus introducing dangerous repetition of process activities); (ii) users cannot be forced to perform any action or task (e.g., they can stand on a page for long time, or even close the browser and disconnect at any time); (iii) the Web architecture itself provides by definition loosely coupled interactions between peers, which become even more uncontrollable in case of Web service-based conversations.

We provide a Web-oriented exception classification, and we represent at a very high level the hypertext structure, the activities performed within this hypertext, and the exceptional situations that may arise; then, we provide a set of policies for capturing exceptions, notifying the user and recovering a regular status of the process. Our approach provides a set of facilities that help the designer through the whole development process of the application: we provide a notation for workflow specification and a framework for defining and classifying exceptions. Capturing mechanisms, user notification patterns and handling policies are defined at a very high level, together with a very basic hypertext model. Then, all this can be deployed in specific hypertext modeling languages (e.g., WebML, as we will show in the last section of the paper), or even directly into implemented code. If a modeling language is selected, the policies can be defined and applied seamlessly, thus allowing automatic code generation too. Note that we disregard aspects such as transactional implications of exceptions or low-level exception handling mechanisms; such issues are not peculiar to Web applications, and therefore can be addressed with traditional approaches.

The paper is organized as follows: Section 2 reviews related work; Section 3 describes the adopted approach, by presenting a running example, a classification paradigm for workflow exceptions on the Web, and the handling policies of exceptions; Section 4 provides an overview of the implementation and validation experiences of the approach, based on the Web Modeling Language; Section 5 draws some conclusions and presents future works.

2. RELATED WORK

Many works have addressed the problem of exception discovery and compensation. They mainly studied transactional properties for activities, which is not in our scope. However, some works deal with weaker properties. For example, [9] is based on the concept of spheres, to make use of only those transactional properties that are actually needed; [13] is one of the first works that addresses the problem in the Web context, but it provides only a classification of exceptions.

For the definition of the scope and the expressive power of workflow primitives, our work is inspired by pattern based workflow analysis by Van Der Aalst [15], and by industrial and academic standards like BPML /BPMN [3] and YAWL [16]. Other works on activity composition and coordination related to the Web have been considered; for example, [4] and others propose solutions on Web services interaction, for which we will propose some exception handling techniques.

As already anticipated, our implementation experience is based on WebML [5][17], a high-level modeling language for data-intensive Web applications, and its extension to workflow-based applications. In [11] an approach for the specification and the design of workflow-driven hypertext suitable for lightweight applications has been proposed: workflow-driven Web applications are defined as hypertexts delivering Web interfaces that permit the execution of activities and embody simple constraints that drive the navigation of users (complex constraint such as complex temporal conditions, composite workflows consisting of independent sub-workflows, and so on are not considered). In this paper, we focus on exceptional behaviors, i.e., behaviors that deviate from the process designed by the analyst.

3. THE PROPOSED APPROACH

The management and recovery of exceptions in a workflow-based Web application require first to identify the typical failures in such a context. In this section we therefore provide a characterization of the exceptions that may arise in a Web application and illustrate the extensions of a process-driven application at a conceptual level.

3.1 The Case Study

In the sequel, we will exemplify the proposed approach on a case study consisting of a Web application implementing a business process. For specifying business processes, we used the Workflow Management Coalition terminology [19] and the BPML/BPMN [3] notation. The workflow model is hence based on the concepts of Process (the description of the business process), Case (a process instance), Activity (the elementary unit of work composing a process), Activity instance (an instantiation of an activity within a case), Actor (a user role intervening in the process), Event (some punctual situation that happens in a case) and Constraint (logical precedence among activities and rules enabling activities execution). Processes can be internally structured using a variety of constructs: sequences of activities, gateways implementing AND, OR, XOR splits, respectively realizing splits into independent, alternative and exclusive threads; gateways implementing joins, a convergence point of more activities; activity iterations; and pre- and post-conditions of activities.

The workflow depicted in Figure 1 describes a loan brokering Web application, providing users with the possibility to search for available loan options, fill loan applications, and be notified of their acceptance/rejection by the loan provider. The desired Web application should cover the whole process, by allowing the broker agent to evaluate the loan request, by giving a preliminary validation, and then checking the details of loan applications (such as the applicants' financial status and job history), and finally registering their final decision on approving or rejecting a given application. If the request is approved, the customer can accept it and proceed to the loan cashing and periodic installment payments.

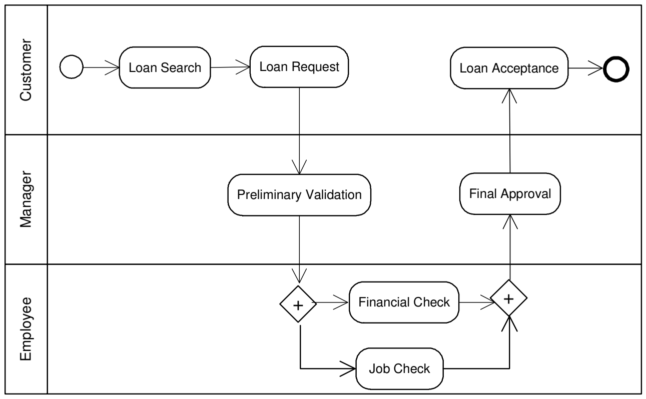

Figure 1. Workflow modeling of the "Loan Request" process

The case study is a simplified process executed through the Web by three actors: customer, brokering manager, and company employee. However, particular situations may be raised during the normal execution of the process and lead the system to an uncertain state. For example:

(a) Within the Loan Request activity, the user may press the *back* button of the browser, thus reaching the first page of the activity, and may try to navigate a link repeating part of the

Request. In this case, a request may be erroneously registered twice.

(b) Within one of the activities, the user may stand on a page for a long time after which a timeout may occur; thus, the user session ends.

(c) A discount rate variation may require changes in the loan conditions of a request under evaluation.

These and other possible exceptions will be better detailed in the next sections.

3.2 Exception Classification

In order to handle exceptions, we try to clarify the conditions under which failures occur. Once an exception is known for the system, handling mechanisms can take place. In the current work, non-identified exceptions cannot be handled; therefore, they leave the process in an uncertain state. The recognized exceptions for a process are classified into three categories:

1. Behavioral (or user-generated) exceptions are driven by the improper execution order of process activities. The free user navigation through Web pages results in the client visiting older pages, trying to explore the hypertext of activities that have already been executed. The following situations can be foreseen: (a) the user, with the *back* button of the browser, visits a page of a *completed* activity and tries to repeat part of the activity or restarts its execution, (b) the user exits the current activity without completing it, either by following a hypertext link or with the *back* button of the browser, (c) the user, with the *back* button of the browser, visits an older page of the *current* activity, thus repeating part of the activity.
2. Semantic (or application) exceptions are driven by the unsuccessful logical outcome of activities execution. For example, the user does not keep paying his periodic installments, or a discount rate variation requires changing the loan conditions.
3. System exceptions are caused by the malfunctioning of the workflow-based Web application infrastructure both at server and at client side. Events that result in such exceptions are network failures and system breakdowns. In the current work, we do not consider server-side failures, since proposals for such an exception context already exist for traditional data-storage and workflow technology. Client-side exceptions are caused either by data storage and client breakdowns, browser crashes, and network unavailability or by elapsed time between user interactions within a process. A client-side failure results either in the client not sending a request to the server, or in the server not responding to the client. Client-side failures are indistinguishable at application level and are recognized as Session End exceptions.

Another important characterization of the exceptions in process-centric Web applications is related to the detection time: exception may occur while the users involved in the process are navigating through the activities of the workflow, or while they are visiting pages not belonging to the workflow, or, possibly, when they are disconnected. Exceptions can be therefore classified as synchronous or asynchronous as follows:

1. Synchronous exceptions occur within an activity of the process, when a page representing the interface for that activity is requested to the server, or more in general, when the user clicks on a link within an activity. In this case, the user session is on, and therefore the exception can be immediately handled.
2. Asynchronous exceptions occur at any time during the process execution, independently of the state of activities in the case. In this case, the user session may be still on (and thus the user may be warned) or may be off (and the exception handling may be deferred).

According to these definitions, behavioral exceptions are always synchronous, because they occur during the navigation of an activity; semantic exceptions may be either synchronous or asynchronous; system exceptions may occur any time during the process execution (for example, when the user disconnects) and are therefore asynchronous.

The following subsections illustrate different exception handling policies that take into account the above classifications.

3.3 Hypertext Modeling

To study in a simple and effective way the exception handling, we introduce a simplified model describing the structure of activities inside hypertexts.

The hypertext belonging to an activity is broken down into pages. Pages are univocally identified within an activity. As shown in Figure 2, the Loan Search activity in the Loan Request process is composed of one hypertext page, identified as page 1. In analogous way, within the Loan Request activity, the Available Loans page is identified with 1, the Loan Details page with 2, and the Loan Ack page with 3. This hypertext allows the user to search for a loan with specific characteristics (Search Criteria page), to look at the results (Available Loans) and to see the details of a specific loan (Loan Details). Then, if the loan is interesting for the user, he can submit a request for that type of loan, and receive a confirmation (Loan Ack page).

Between two subsequent pages, there can be a chain of operations executed at server-side (depicted as small circles in Figure 2), which are not relevant for our purposes. Indeed, since we do not consider server-side failures, a chain of operations can be seen as an atomic element that never fails. Thus, within the representation of process hypertexts we do not consider server-side operations.

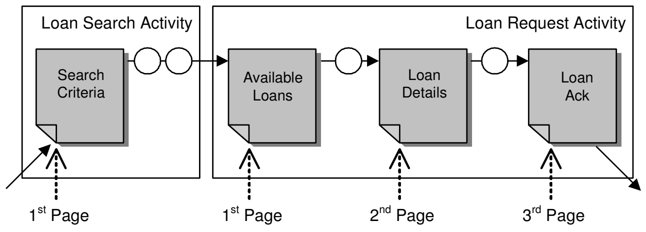

Figure 2. Process hypertext modeling presenting server side operations and pages numbering

Another feature of workflow-based Web applications that we take into account is the possible invocation of Web Services within an activity, aiming at integrating remote services or data within the user browsable hypertext. Thus, between two subsequent pages, there can be a Web Service call whose unsuccessful execution can raise critical situations. In this case, the previous hypertext model is extended considering the Web Service interaction as a numbered page within the activity. In the rest of the paper, we assume that the Loan Search activity is actually implemented with a Web Service call to a service which is outside of the bank offering the loans. Therefore, the Loan Search activity is composed of two steps as shown in Figure 3.

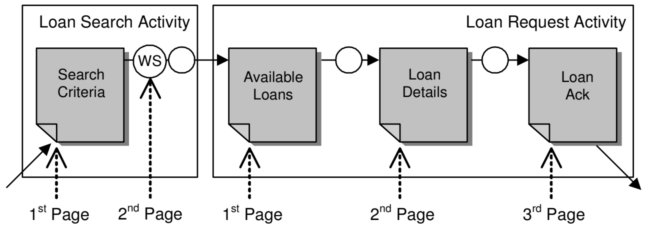

Figure 3. Process hypertext modeling integrated with Web Services interaction

3.4 Workflow Metadata Modeling

Managing exceptions in workflow-driven Web applications requires the storage and retrieval of state information about workflows and exceptions. Therefore, we introduce the data model that includes application data together with workflow and exception metadata. Figure 4 represents the data model of the running example. Application data, following the Entity – Relationship model, is composed of entities describing domain objects and relationships among them imposing data connections. In our example, application data consists of *LoanProposals* whose conditions are imposed from a *Country*; each loan proposal can be fulfilled based on various *InstallmentPlans*; the customer may choose one of these plans and submit a *LoanRequest*, and then, if the request is approved, he can proceed to the periodic payment of the instalment plans, which are recorded by the *InstPayment* entity.

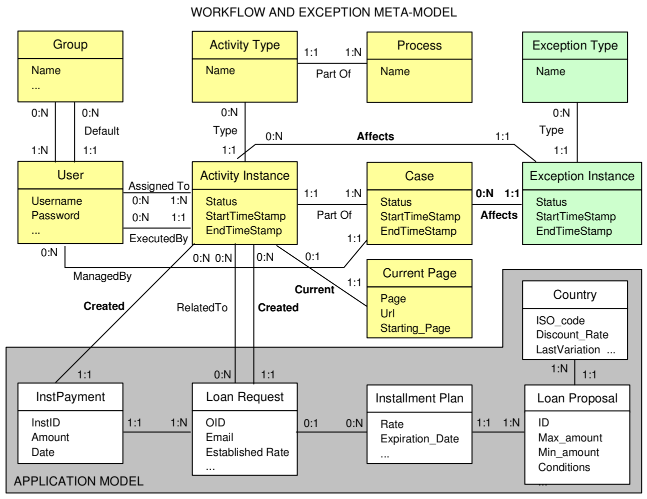

Figure 4. Data model incorporating workflow and exception information

In the E-R model of Figure 4, the upper part contains the entities representing the basic concepts of a workflow, both at type level (Process, ActivityType, and Group) and at instance level (Case, ActivityInstance, and User). This metadata is inspired by the WFMC specification [19], with entities describing the elements of a process and relationships representing the semantic connections between these elements. Entity *Process* is associated with entity *ActivityType*, to represent the classes of activities that can be executed in a process. Both entities describe general data about processes and activities, which need not be replicated for each process/activity instantiation. Entity *Case* denotes an instance of a process, which has a name, used as a label for communicating with the user, a start time, an end time and a status (initiated, active when at least an activity is started, or completed). Entity *ActivityInstance* denotes the occurrence of an activity, described by a start time, an end time and a status which (inactive, active or completed). Entities *User* and *Group* represent the workflow actors, as individual users organized within groups. Activities instances are "assigned to" one or more users who can perform the activity, but are actually "executed by" a single user. The "RelatedTo" connection is required to connect the workflow activities to the data items they use. In this way, it is always possible to deduce from any application object the activity instance(s) where the object is currently in use (through the RelatedTo connection), and, consequently, the case(s) and the user(s) associated with the activity instance (through the PartOf, ExecutedBy and AssignedTo connections). This model is sufficient to master the regular flow of processes, i.e., a flow without exceptions [5].

Exception execution context is captured in Figure 4 by the exception metadata. Entity *Exception* denotes the classes of known exceptions that may occur during the process execution and are described by a name. The actual occurrence of an exception is presented in the entity *Exception Instance* and is described by start time, end time and status which can be: active (i.e., the exception has occurred and has not been addressed yet), resolving (i.e., the exception is currently being solved by a predefined policy or a compensation chain) and resolved (i.e., the exception has been solved by some exception handling mechanism).

The above exception modeling is not enough to describe critical situations within a process execution. A mapping is also necessary in order to relate the generated exceptions to workflow concepts. For example, a variation in a country's discount rate is an exception that needs to be modeled for every process that has requested a loan whose conditions are imposed from the specific country. A behavioral exception within the Loan Request activity needs to be mapped to the corresponding activity instance. These mappings are modeled by the data connections *Affects* between the *Exception Instance* and *Activity Instance* / *Case* entities. In this way, it is always possible to retrieve the exceptions that are raised for an activity instance and (or) a case, and vice-versa.

Further objects in the exception metadata are introduced for exception handling. The executor of the appropriate exception handler is either the user who has actually performed the affected activity instance and is recognized by the "Executed By" connection, or the user denoted by the "Managed By" connection as the responsible to handle exceptions for the specific case.

We define the *current page* of an active activity as the last page that the server has generated after a request by the client. This information is stored into the *CurrentPage* entity of the workflow metadata schema. For example, the identification of the current page for an activity in Figure 3 occurs after a user request within the corresponding activity. Within a process case, it is always possible to retrieve the currently active activities, and, for each of them, the current page. The current page has two important properties: (i) it is always uniquely defined for an active activity, and (ii) it gives us correct idea of the progress of the activity. Even if the client uses the back and forward buttons of the browser, the current page of the activity does not change, since the client does not make any request to the server. Moreover, by clicking the *back* button the system does not roll back the operations between consecutive pages, it just reloads an old page.

Finally, the last object useful for applying recovery mechanisms is the *Created* relationship that connects the Activity Instance to the application data object that is managed within the workflow. It keeps track of the activity during which a specific object has been created, at the purpose of allowing possible removals of the object when the activity is cancelled. Handling exceptions occurs at the price of managing such extra-information, which is a sort of process log.

3.5 Handling Approach

In this section, we show how the workflow meta-data introduced in Section 3.4 can be used to manage exceptions. Our exception handling proposal is based on three models: (1) the capturing model, used to capture events and to store the exceptions data in the workflow meta-data model, (2) the notifying model, used to notify the occurred exceptions to the user inside the hypertext model, and (3) the handling model, used to resolve the exceptions, by applying a recovery policy. In this scenario, the exception management remains apart from the normal workflow design.

The capturing model incorporates all the mechanisms used to capture events and generate exceptions in the workflow meta-model. We propose two different mechanisms for the exceptions classified in Section 3.2, namely, a) triggers, used for capturing exceptions caused by data modifications: b) Web services, used for capturing exceptions explicitly notified by external sources (or, possibly, by the application itself). Figure 5 summarizes how these mechanisms can be used by the different kinds of exceptions.

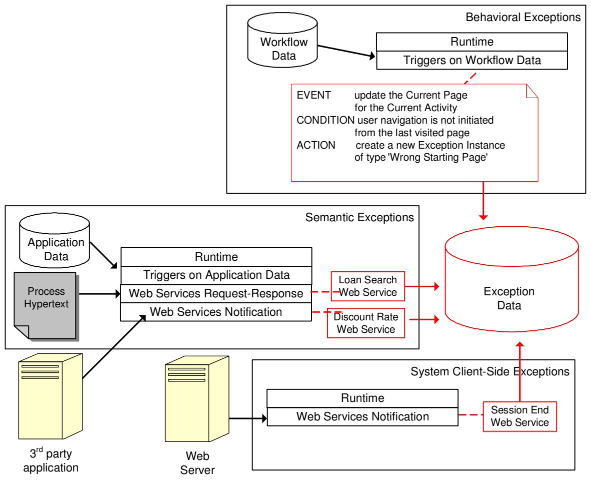
Behavioral Exceptions
Workflow Data
Runtime
Triggers on Workflow Data
EVENT update the Current Page for the Current Activity
CONDITION user navigation is not initiated from the last visited page
ACTION create a new Exception Instance of type 'Wrong Starting Page'
Semantic Exceptions
Application Data
Process Hypertext
Runtime
Triggers on Application Data
Web Services Request-Response
Web Services Notification
Loan Search Web Service
Discount Rate Web Service
Exception Data
System Client-Side Exceptions
Runtime
Web Services Notification
Session End Web Service
3rd party application
Web Server

Figure 5. Capture Model for Behavioral, Semantic and System client-side exceptions

Behavioral exceptions can be captured by the application by means of triggers defined on the workflow meta-data. For example, Figure 5 shows the trigger for the generation of the behavioral exception “Wrong Starting Page”, raised when the user reaches a page preceding the actual current page of the workflow, thus trying to repeat part of the same activity or an already completed activity. The *event* part of the trigger captures the exception event: the update of the current page for the current activity in the workflow meta-data. This update is performed every time a client request is made within an activity in order to identify the last page the server has generated for the activity. The *condition* part checks if the current client request does not come from the page identified as the actual current page in the database. If the condition holds, the *action* part of the trigger generates a new exception instance of type “Wrong Starting Page” and connects it to the current activity instance.

Semantic exceptions may instead be notified by external sources, and therefore be captured through Web services, as reported in Figure 5[1]. Web services capture the two following kinds of exceptions: 1) an exceptional response received after calling a Web service (for example, if the loan search activity calls a Web Service to get the loans satisfying the criteria specified by the user, and the amount inserted by the user exceeds the limits for which the loan can be approved, a failure in the response may be raised); 2) an exception notified by an external application by means of a notification Web service (for example, the notification of the discount rate variation for a country). In both cases the Web services are responsible to generate the new exception instances and to connect them to the corresponding activity instances or cases. Semantic exceptions can also be captured by means of triggers defined on the application data. As an example, consider the modification of the expiration date of an installment plan. This event affects all the cases where the requested loan fulfilled such installment plan. In this case a trigger can be specified to generate a new exception instance of type “Expiration Date Modification” for every affected case and to connect it to the corresponding cases.

System (client-side) exceptions result in a communication failure between the client and the server. In such situations, the client unavailability is not immediately detected from the Web Server: such detection occurs after a timeout or after an unsuccessful communication from the server-side. Also the notification of such failures can be done by means of Web Services (see Figure 5). When the Web Server detects the communication failure with the client, either because it cannot send the server’s response or because the client does not make a request for a specific amount of time, it calls the Session Web Service published at server side, with the necessary parameters, like the identifiers of the activities that were in progress and have been interrupted. The Session End Web Service generates a new exception instance of type “Session End” and connects it to the corresponding activity instances.

[1] Our choice of using Web services as opposed to other inter-process communication mechanisms is motivated by our focus on interoperability and portability. Web services are published by the Exception Manager and are invoked either by the workflow application itself or by external applications.

The capturing model presented above can be designed separately from the process modeling. In the following, we will see how the generated exceptions are utilized and therefore integrated in the normal process flow in order to be handled.

The notifying model incorporates all the mechanisms used to present the captured exception (stored into the database) to the user. Synchronous and asynchronous exceptions require different mechanisms. Indeed, if the exception occurs synchronously during the execution of the workflow activities, it can be notified immediately inside the affected activity; otherwise, the notification should take place outside the process flow. To notify the exceptions the process hypertext modeling introduced in Section 2.3 is extended with new primitives, allowing the detection of exceptions in the workflow meta-data: in particular, in order to support synchronous exceptions the concept of *exception-aware link* is introduced, which is a navigational link extended with the ability to check the exceptions occurrence in the database and to redirect the process flow to the recovery mechanisms; in order to support asynchronous exceptions an *exception-control* mechanism is defined to be included in a generic page: it activates a hypertext link to the recovery mechanism in case of exception occurrence. These two mechanisms are illustrated in Figure 6 and Figure 7.

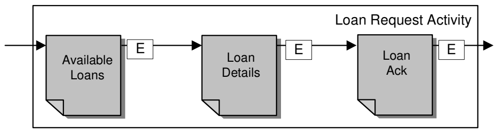

Figure 6. Notification of the user in mode synchronous

Figure 6 demonstrates the mechanism of notification for synchronous exceptions; i.e., exceptions generated after a user request within the activity execution. Hypertext links within the activity flow can be marked as *exception-aware links* (graphically represented with the “E” label). The navigation of these links fires an automatic control to the database, checking the occurrence of specific exceptions for the current activity. If such an exception has occurred, the link leads the user to a recovery process, presented in the following subsection. For example, the Exception-aware links outgoing from the pages “Available Loans”, “Loan Details” and “Loan Ack” in the Loan Request activity are defined (at design time) to check (at navigation time) the occurrence of the behavioral exceptions for the activity.

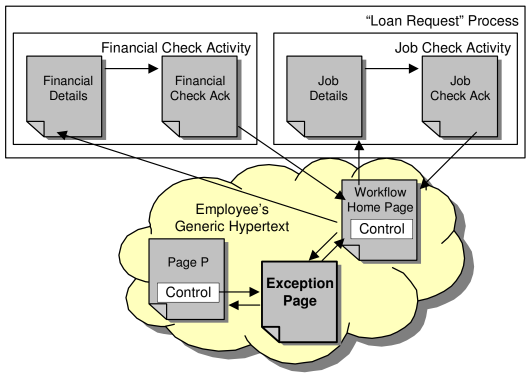

Figure 7. Notification of the user in mode asynchronous

Figure 7 shows an example of user notification in case of asynchronous exceptions; in this case, exceptions may occur independently of the process execution but are generated for activities executed by the current user or for cases managed by the current user. In the hypertext design an Exception Control primitive can be inserted inside normal hypertext pages (e.g., in the Workflow Home Page or in Page P in the figure) to control exceptions occurrence before the page is loaded and to activate a hypertext link if such exceptions have been generated in the database. Navigating this link, the user is led to an Exception page where information about the above exceptions is retrieved.

The handling model represents the mechanisms used to recover the exceptions. It consists in recovery operations that take place on the affected activities or cases in order to bring the application to a consistent state, so that the process execution can proceed. The recovery policies can be either predefined or user-defined. *Predefined policies* are server-side operations that receive initial input parameters about the exception to resolve.

They are automated mechanisms that can be applied to different exception types. The process execution, after the exception is handled, is implicitly specified and routed from the predefined policy. We have identified five predefined policies:

(a) The *Accept* policy: it accepts all the operations done by the affected activity/case and concludes the activity/case execution by setting its status to Completed.

(b) The *Reject* policy: it deletes all the data created by the affected activity/case and enables its re-execution by setting its status to Inactive.

(c) The *Abort* policy: it accepts all the operations done by the affected activity/case and concludes the activity/case execution by setting its status to Aborted.

(d) The *Ignore* policy: it informs the user of the occurred exception with a message and resumes the flow execution.

(e) The *Resume* policy: it resumes the user navigation from the last visited page generated by the server for the affected activity; all the data created by the affected activity/case after the last visited page navigation are deleted.

User-defined policies are defined by the application designer to manage critical situations when automated mechanisms cannot restore the process state. For example, the exception caused by the discount rate modification may require the revision of the conditions of the loan proposals: such variations need to be explicitly modeled by the designer. User-defined policies specify: (i) the pages/operations to be executed to handle the exception and (ii) how to continue the process execution after the exception handling.

Figure 8 and Figure 9 illustrate two examples of specification of the handling model to recover a synchronous and an asynchronous exception, respectively. Recovery policies are modeled as operations (graphically depicted as circles). Dotted curved lines represent the calling of an exception handling mechanism and the returning to the normal workflow.

The exception handling depicted in Figure 8 represents a possible runtime scenario for an exception caused by the improper user navigation within the Loan Request activity. Suppose, for example, that the user has visited the last page of the activity, "Loan Ack" and that the loan request details are registered in the database (a). At this point, the only enabled link is the one exiting the page. By navigating this link, the activity is completed. If the user does not follow this link but presses the *back* button of the browser twice, he reloads the already visited page "Available Loans" and can therefore navigate the outgoing link of that page (b). This navigation is automatically followed by the modification of the Current Page in the exception meta-data, which fires the corresponding trigger. A new exception instance of type "Wrong Starting Page" is raised for the current activity. The Exception-Aware Link is defined to check the occurrence of behavioral exceptions related to the current activity/case for the user navigating this link. If the "Wrong Starting Page" exception is recognized, the exception-aware link leads the user to the corresponding recovery policy. In this example, the Resume policy is automatically applied. As a consequence, the user navigation is resumed from the last visited page, that is the "Loan Ack" page (c).

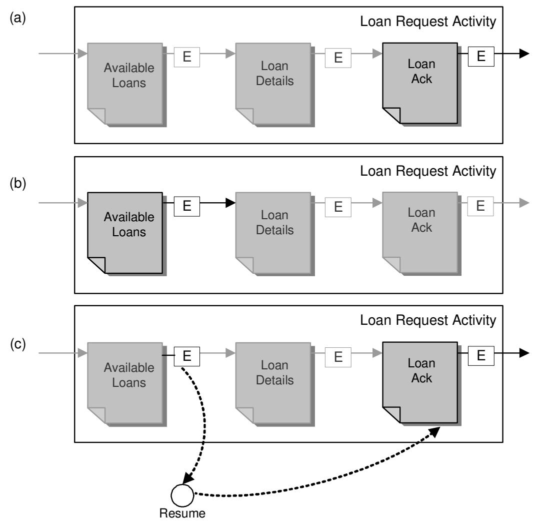

Figure 8. "Wrong Starting Page" handling in mode synchronous for the "Loan Request" activity

Figure 9 presents an example of the exception handling for an asynchronous exception of type "Session End", caused by the delayed user navigation within the "Financial Check" activity. In particular, the following situation is described: the employee has visited the second page of the activity, "Financial Check Ack". In the database, the financial control details are registered in the Loan Request entity and the Current Page denotes the "Financial Check Ack" page as the last one loaded by the server page for the current activity. For 10 minutes, the user does not navigate the outgoing link of that page; thus, a timeout defined in the Web Server expires, the Session End Web Service is called from the Web Server and the corresponding exception is stored into the database (a). When the user finally tries to navigate to the next page of the activity, since the session has expired he is redirected (from the Web Server) to the login page (b). After the successful login, the Workflow Home Page is requested to the server. Before loading this page, the Exception Control unit checks if exceptions have been stored in the database for the user, and activates the link to the "Exception" page. From the "Exception" page the user may choose to Resume or Reject the "Session End" exception for the Financial Check activity. If he chooses the Resume policy, then he is redirected to the last loaded page before the exception, the "Financial Check Ack" page, and the process execution may proceed. If he chooses the Reject policy, the system tries to recover the initial state of the database before the activity execution; thus, it deletes the financial control details from the Loan Request entity and assigns the Inactive status to the activity so that the activity may be re-executed. The user is then transferred to the Home Page.

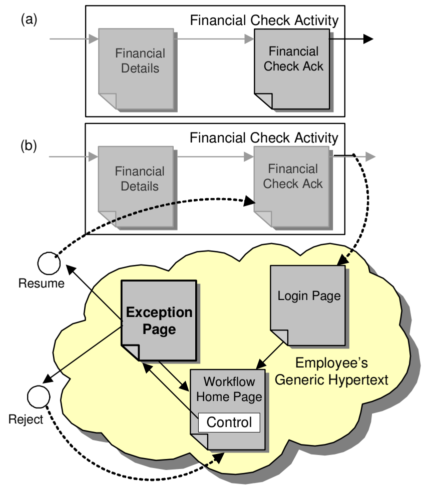

Figure 9. "Session End" Handling in mode asynchronous for the "Financial Check" activity

4. IMPLEMENTATION AND EXPERIENCE

The approach for the exception handling illustrated in Section 2 has been specified using the WebML language [5][17], a high-level notation for data- and process-centric Web applications, and has been implemented in a prototype that extends the CASE tool WebRatio [18], a development environment for the visual specification of applications in WebML and the automatic generation of code for the J2EE and Microsoft .NET platforms. After a brief overview of WebML we present the main extensions of the language needed to specify the exceptions and the recovery policies, and show some typical usage patterns.

4.1 A Brief Overview of WebML

WebML allows specifying a Web site on top of a database. Such a conceptual Web specification consists of a *data schema,* describing application data, and of one or more *hypertexts*, expressing the Web interface used to publish this data.

The WebML data model is the standard Entity-Relationship (E-R) model, widely used in general-purpose design tools. Upon the same data model, it is possible to define different hypertexts (e.g., for different types of users or for different publishing devices), called *site views*. A site view is a graph of *pages*, allowing users from the corresponding group to perform their specific activities. Pages consist of connected *units*, representing at a conceptual level atomic pieces of homogeneous information to be published: the content that a unit displays is extracted from an entity, and selected by means of a *selector*, testing complex logical conditions over the unit's entity. Units within a Web site are often related to each other thru *links* carrying data from a unit to another, to allow the computation of the hypertext.

WebML allows specifying also update *operations* on the data underlying the site too (e.g., the creation/deletion of instances of an entity, or the creation and deletion of instances of a relationship) or operations performing other actions (e.g. send an e-mail). In [5] the language has been extended with operations supporting process specifications (but not exception handling), while in [11] also Web service calls and specifications have been included.

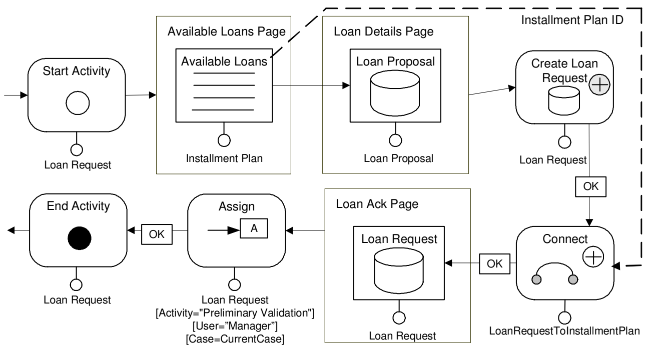

Figure 10. WebML implementation of the "Loan Request" activity

Figure 10 shows a fragment of hypertext specified in WebML. A *start activity operation* starts the Loan Request activity, setting its state to "active". Then, the Available Loans Page is shown to the user: it contains an *index unit* showing all the available loans matching the search criteria of the previous activity. When the user selects a loan, the Loan Details page shows the details of the selected loan to the user (Loan Proposal *Data unit*). If the user chooses to submit a request for the selected loan type, a *create operation* inserts a new instance in the Loan Request table and a *connect operations,* creates the relationships between this new instance and the LoanRequestToInstallment relation. In the next Loan Ack page, the confirmation details of the newly created requested loan are shown to the user by the Loan Request *Data unit*. The activity ends with the assignment (*assign operation*) of the newly created instance to the manager and to the Preliminary Validation activity, which is therefore enabled for its execution, and with the *End activity operation*, which sets the status of the Loan Request activity to "completed".

The language is *extensible*, allowing for the definition of customized operations and units, implemented in the WebRatio CASE tool as plug-ins. In our prototype, the language has been therefore extended with new primitives to support exceptions. For a complete description of the language and of the architecture supporting WebML the reader may refer to [5][17].

4.2 Extensions of WebML for Supporting Workflow Exception Handling

The specification of the exceptions in WebML and the implementation of all the policies require the following extensions.

Workflow metadata modelling. The data model of the Web application has been extended with the workflow and exception meta-models as described in Section 2.4.

Capturing model. Exceptions captured using the triggering mechanisms have been directly specified in the underlying database management system. Exceptions captured using the Web service mechanisms are instead specified using the WebML extensions for Web services, illustrated in [11]. Figure 11 shows some examples of WebML specification for capturing exceptions.

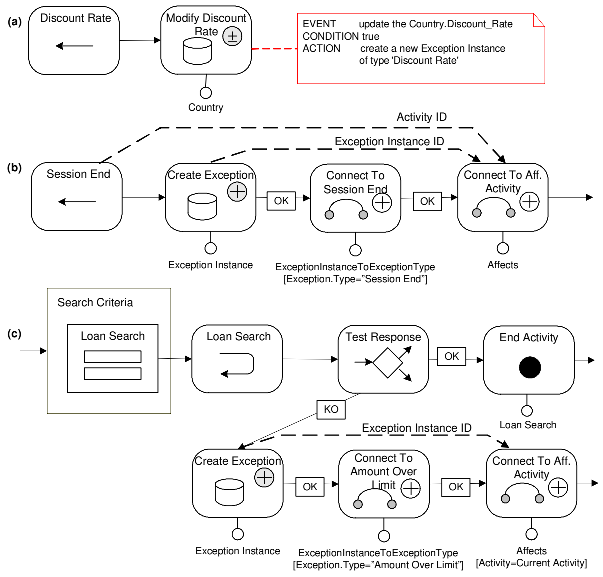

Figure 11. Capturing an external exception with a Web service

The hypertext fragment (a) shows the capturing mechanism of a semantic asynchronous exception coming from an external application, representing the variation of the discount rate for a country. A notification service operation (called Discount Rate) is published. When it is called, it performs the modification of the Discount Rate attribute in the Country entity of the application data. This modification fires then an existing trigger, whose action part generates a new exception instance of type "Discount Rate" and connects it to the all the cases that have requested a loan whose conditions are imposed from the specific country.

The hypertext fragment (b) in Figure 11 shows the WebML implementation for capturing Session End exceptions. Even in this case, a notification service operation (called Session End) is published. When it is called, a create operation inserts a new instance in the *Exception Instance* entity, connects the newly created instance with the exception type "Session End" (thus creating a new instance of the relationship *Type*), and with the entity representing the affected activity (thus creating a new instance of the *Affects* relationship).

The hypertext fragment (c) in Figure 11 shows the WebML implementation for capturing the semantic synchronous exception representing the failure of the loan search based on customer's criteria because the requested loan amount exceeds the supported limit. A request-response operation (Loan Search) is published and called within the Loan Search activity. If the response message indicates the failure, the following operations apply; a new exception instance is created, and connected through the Type and Affects relationships to the "Amount Over Limit" Exception Type, and to the instance of the current activity. After capturing the event, the recovery of the exception will be then specified.

Notifying and handling models. These two models have been implemented by adding the following new primitives to the WebML language:

1. *Exception control unit*: checks whether an exception has occurred either for an activity executed by the current user, or for a case managed by the current user.

2. *Exception-aware content unit*: retrieves the activities instances or the cases affected by an exception of a specific type occurred for the current user and shows also the recovery policies associated (at design-time) with that kind of exception.

3. *Recovery units* for implementing the five recovery policies described in Section 2.5 (accept, reject, abort, ignore, resume): such units can automatically handle the exception to recover. Indeed, as described in the exception meta-data model in Section 2.4, given a specific exception instance it is possible to retrieve the affected activity instance and/or the corresponding case, all the objects created within the activity, all the objects assigned to that activity, and the current page visited by the user within that activity. All these data allow an automatic implementation of all the policies.

4. A set of units for specifying *user-defined handlers*: in particular, the operations to be performed by the handler are enclosed between a *start* and an *end exception handler* unit (like the start and the end activity operations shown Figure 10), and when all the operations have been performed a Goto-page unit allows to specify the page in the hypertext from which the navigation must continue (e.g., the current page, the first page of a particular activity, the home page, and so on).

These units can be considered as "macros" performing suitable queries and updates on the exception subschema of the meta-model. In addition, WebML supports the new notion of exception-aware link, as discussed in Section 3.5. Notice that with the help of these units the designer should not directly access exception data; he must specify where exceptions should be checked and then concentrate only on the recovery policies to adopt for each kind of exception. For their specification, some typical *patterns* have been identified. Here we show two examples, one applied to a synchronous exception and one applied to an asynchronous exception.

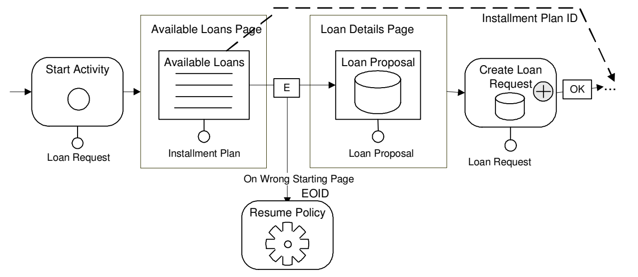

Figure 12. Example of pattern for a synchronous exception

Figure 12 presents the typical pattern for a synchronous exception: it implements the example of the behavioral exception "Wrong Starting Page" discussed in Section 3.5 and illustrated in Figure 8. This exception is captured by a trigger (see Figure 5). In the Available Loans Page, the customer receives a loans list (represented by an index unit) matching his search criteria. When the customer clicks on a link of the index the exception-aware link labeled with "E" is followed, which checks if the "Wrong Starting Page" exception has occurred. In such case, the Resume Policy, represented by the corresponding recovery unit, is applied; otherwise, the activity continues.

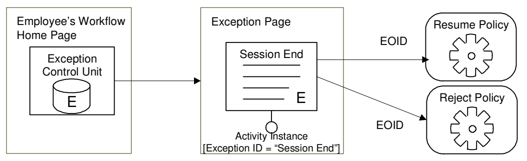

Figure 13. Example of pattern for an asynchronous exception

Figure 13 depicts the typical pattern for an asynchronous exception: the "Session End" exception, discussed in Section 2.5 and illustrated in Figure 9. This exception is captured by means of a Web service (see Figure 11.b). In this case, the exception is not handled within an activity, but in a different page, for example, in the home page of the customer workflow. In this page an exception control unit checks if an asynchronous exception has occurred for activities assigned to the customer or for cases managed by the customer, and activates a link to the corresponding exception handler page. In the Exception Page an exception-aware index unit retrieves all the activities instances affected by a "Session End" exception that can be recovered by the employee and shows also the recovery policies associated with it: in this example, the resume or the reject policies may be chosen by the employee. The selection of a particular policy for a particular activity instance yields to the execution of the corresponding recovery unit.

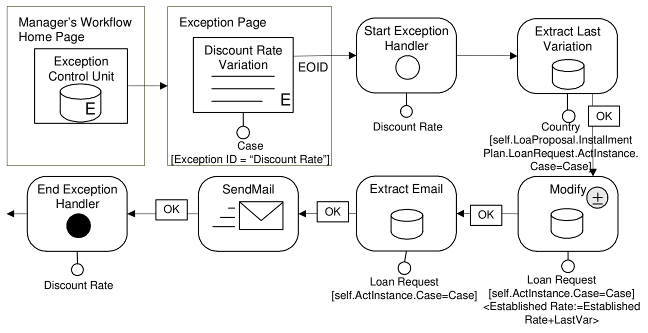
Manager's Workflow Home Page
Exception Control Unit
E
Exception Page
Discount Rate Variation
E
EOID
Case
[Exception ID = "Discount Rate"]
Start Exception Handler
Discount Rate
Extract Last Variation
Country
[self.LoaProposal.Installment Plan.LoanRequest.ActInstance. Case=Case]
OK
Modify
Loan Request
[self.ActInstance.Case=Case]
<Established Rate:=Established Rate+LastVar>
OK
Extract Email
Loan Request
[self.ActInstance.Case=Case]
OK
SendMail
OK
End Exception Handler
Discount Rate

Figure 14. Example of user-defined policy

Finally, Figure 14 shows an example of user-defined policy, recovering the discount rate variation exception. In our simplified example, we suppose that the manager is asked to update the affected loan requests by adding the LastVariation to the Established Rate attribute of the Loan Request related to an Installment Plan of the interested Country. More precisely, once the exception is recognized: (i) the Cases related to the exception are retrieved through an Exception index unit; (ii) the manager chooses a case and the handling policy is started; (iii) the Established Rate of the Loan Request related to the selected case is updated; (iv) an email message is sent to the customer of the Loan Request; and finally (v) the handling policy is closed.

4.3 Experience

The concepts presented in this paper have been proved valid on the field. Several case studies exploiting exception handling capabilities have been implemented, thus validating and refining the approach. The most relevant applications include the Acer Business Portal, an application invoking and defining remote service calls for providing location and driving information to users, and supporting workflow-based interaction between Acer and its commercial partners; and MetalC [12], a complex application including a set of B2B portals, one for each business partner. The purpose of this second project is to allow business interactions between small Italian companies of the mechanical sector by means of their respective Web portals, through Web services calls. In this context, complex workflow interactions have been put in place, to grant reliable cooperation. For example, the purchasing process in a B2B scenario consist of a very complex set of interactions, since the buyer typically asks for a quote, the seller makes his offer, then the buyer sends his order for the best offer. In this context, exceptions management becomes very critical. In the implemented communication platform all the discussed recovery policies have been used. Some examples follows: (i) if an exception occurs within the AskForQuote activity, an accept policy is synchronously performed, and the request is sent even if not all the data are submitted (less relevant data are left in the last steps of the activity); (ii) if an exception occurs within the SendOrder activity, the reject policy applies in synchronous modality: data created within the activity is deleted, and the user is asked to restart it; (iii) in case of exception within the self-registration activity, which is a long sequence of data submission by the partners, resume policy is exploited, to allow the user resume the self-registration from the point in which he left the application. An example of user-defined recovery becomes necessary within the shipping confirmation activity: once the order has been confirmed and the goods are ready to be shipped, the seller must notify the buyer about the sending. If an exception occurs during the execution of this activity, a user-defined compensation chain is performed, automatically executing the remaining steps of the activity. Finally, also some asynchronous exceptions have been defined, like for example the "Session End" exception.

5. CONCLUSIONS

In this paper, we have proposed a conceptual approach to exception handling within workflow-based Web applications. We start with a formal classification of exceptions that can specifically occur during the execution of Web applications. This classification allows a clear identification of the possible sources of critical situations, and provides simple guidelines for designing appropriate solutions for the various scenarios. The Web application is modeled with a very high level representation, comprising a metadata model, a graph of activities made of pages, and a set of primitives to be used into hypertext specification. Thus, the main advantage of our approach stands in allowing the definition of exception handling policies and compensation chains without lowering the abstraction level of the design.

Once the policies and the high-level design are exploited, it is possible to move to a more detailed design based on traditional modeling languages (and therefore exploiting automatic generation of the code), or directly on the implementation level. Future work will include further implementation experiences, to allow the refinement of our approach.

6. REFERENCES

[1] Atzeni, P., Mecca, G., Merialdo, P.: Design and Maintenance of Data-Intensive Web Sites. EDBT 1998, 436-450.

[2] Baresi, L., Garzotto, F., Paolini, P.: From Web Sites to Web Applications: New Issues for Conceptual Modeling. ER Workshops 2000, 89-100.

[3] BPML and BPMN site http://www.bpmi.org/.

[4] Bultan, T., Fu, X., Hull, R., Su, J. : Conversation specification: a new approach to design and analysis of e-service composition. WWW 2003, 403-410.

[5] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications, Morgan-Kaufmann, 2002.

[6] Conallen, J.: Building Web Applications with UML. Addison Wesley (OTS), 2000.

[7] Fernandez, M. F., Florescu, D., Kang, J., Levy, A. Y., Suciu, D.: Catching the Boat with Strudel: Experiences with a Web-Site Management System. SIGMOD 1998, 414-425.

[8] Gómez, J., Cachero, C., Pastor, O.: Conceptual Modeling of Device-Independent Web Applications. IEEE MultiMedia 8(2), 26-39, 2001.

[9] Hagen, C., Alonso G.: Exception Handling in Workflow Management Systems. IEEE TSE 26(10), 943-958, 2000.

[10] Hennicker, R., Koch, N.: A UML-based Methodology for Hypermedia Design. UML 2000, 410-424.

[11] Manolescu, I., Brambilla, M., Ceri, S., Comai, S., Fraternali, P.: Model-Driven Design and Deployment of Service-Enabled Web Applications, ACM TOIT, Vol. 5(2), May 2005.

[12] MetalC Web site http://www.metalc.it .

[13] Miller, J. A., Sheth, A. P., Kochut, K. J., Luo Z. W.: Recovery Issues in Web-Based Workflow. CAINE-99, Atlanta, Georgia, 101-105, November 1999.

[14] Schwabe, D., Rossi, G.: An Object Oriented Approach to Web Applications Design. TAPOS 4(4), 1998.

[15] Van der Aalst, W. M. P., ter Hofstede, A. H. M., Weske: Business Process Management: A Survey. Business Process Management 2003.

[16] Van der Aalst, W. M. P., Aldred, L., Dumas, M., ter Hofstede, A. H. M..: Design and Implementation of the YAWL system, CAiSE 04, Riga, Latvia, June 2004. Springer Verlag.

[17] WebML Web site http://www.webml.org .

[18] WebRatio site http://www.webratio.com .

[19] Workflow Management Coalition site http://www.wfmc.org