Exception Handling Within Workflow-Based Web Applications

Marco Brambilla and Nicola D'Elia

Dipartimento Elettronica e Informazione, Politecnico di Milano,
Via Ponzio 34/5, 20133 Milano, Italy
mbrambil@elet.polimi.it, nd636136@polimi.it

Abstract. As the Web becomes a platform for implementing B2B applications, the need arises of extending Web conceptual modeling from data-centric applications to data- and process-centric applications. New primitives must be put in place to implement workflows describing business processes. In this context, new problems about process safety arise, due to the loose control on Web clients. Indeed, user behavior can generate dangerous incoherencies for the execution of processes. This paper presents a proposal of workflow-enabling primitives for Web applications, and a high level approach to the management of exceptions that occurs during execution of processes. We present a classification of exceptions that can occur inside workflow-based Web applications, and recovery policies to retrieve coherent status and data after an exception. An implementation experience is briefly presented too.

1 Introduction

In recent years, the Web is more and more being used as the implementation platform for B2B applications, whose goal is not only the navigation of content, but also supporting business processes, content management, value-added services and so on. Conceptual modeling expertise from other fields (database, object-orientated programming, hypermedia applications) has been widely recognized as valid starting point for defining conceptual aids for Web application development too [7]. The first generation of conceptual models for the Web [1, 2, 5, 6] essentially focus on capturing the structure of data to be published, and the navigation primitives, represented by such concepts as pages, content nodes, and links.

To cover business processes support, a second generation of conceptual models is required. These new models should cope with process and workflow modeling, support Web service interaction, and integrate data-centric and process-centric modeling primitives into a mix suited to the development of advanced B2B Web applications. In this context, it is important to address the critical cases that can occur in the enactment of business processes on a Web-based platform.

This paper presents an extension to a first-generation Web modeling language [5, 6] to support the specification, design and implementation of B2B applications, and

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 103–117, 2004.

offers an high-level analysis of critic aspects and exception management issues within Web applications exploiting business processes. Exceptions that can happen in a Web based application have peculiar characteristics with respect to traditional workflow applications. This is due to two main aspects: (i) interaction options provided by browser-based interfaces are very powerful, but they are more oriented to free navigation than strict processes adherence(e.g., user is enabled to jump back and forth on navigated pages, thus introducing dangerous repetition of process activities); (ii) user cannot be forced to perform any action or task (e.g., he can stand on a page for long time, or even close the browser and disconnect at any time).
Our approach is lightweight: we are interested in extending Web modeling to cope with process and exception modeling, not to adapt workflow management systems to the Web; about exceptions, we aim at defining a modeling paradigm for critical cases, not to build transactional systems or low level exception handling mechanisms.
Many works have addressed the problem of exception interception and compensation. They mainly studied transactional properties for activities, which is not in our scope. However, some works deals with weaker properties. For example, [8] is based on the concept of spheres, to make use of only those transactional properties that are actually needed; [12] is one of the first works that address the problem in the Web context.
The paper is organized as follows: Section 2 briefly outlines the main concepts about workflow and Web application modeling; Section 3 introduces the study of exception and critical situations that can occur in the execution of processes on the Web; Section 4 presents our approach to management and recovery of exceptional situations within process execution; finally Section 5 reviews implementation experience and Section 6 draws some conclusions and presents our ongoing and future work.

2 Conceptual Modeling of Web Applications and Workflows

Conceptual design consists in high-level, platform-independent specification of the application, which can be used to drive the subsequent implementation phase. In this section we focus on two aspects of conceptual design: (i) Web application design, briefly describing the WebML model, that will be used in the sequel to describe our proposals; (ii) Workflow modeling concepts and primitives.
It is important to point out that, although the paper uses the WebML notation to describe our contribution, the proposed approach is independent from the specific language or notation that is adopted. Our approach to conceptual design relies on the following guidelines: an Entity-Relationship diagram models the data stored, manipulated, and exchanged by the application actors, plus the metadata required for the management of the business processes; *process diagrams* are treated as a higher-level specification and are used to derive a set of hypertext models that "realizes" them. These hypertext models belong to the site views of the user groups involved in the process and must offer them the interface needed for performing their activities.

2.1 Process Modeling

For specifying processes, we adopt the terminology and notation defined by the Workflow Management Coalition [16], which provides a workflow model based on the concepts of *Process* (the description of the supported workflow), *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), and *Constraint* (logical precedence among activities and rules enabling activities execution). Processes can be internally structured using a variety of constructs: sequences of activities, AND-splits (a single thread of control splits into two or more independent threads), AND-joins (blocking convergence point of two or more parallel activities), OR-splits (point in which one among multiple alternative branches is taken), OR-joins (non-blocking convergence point), iterations for repeating the execution of one or more activities, pre- and post-conditions (entry and exit criteria to and from a particular activity).
Fig. 1 exemplifies a WfMC workflow specifying the process of online purchase, payment and delivery of goods. The customer can choose the products to purchase, then submits his payment information. At this point, two parallel tasks are executed by the seller employees: the warehouse manager registers the shipping of the order, and a secretary prepares a bill to be sent to the customer.

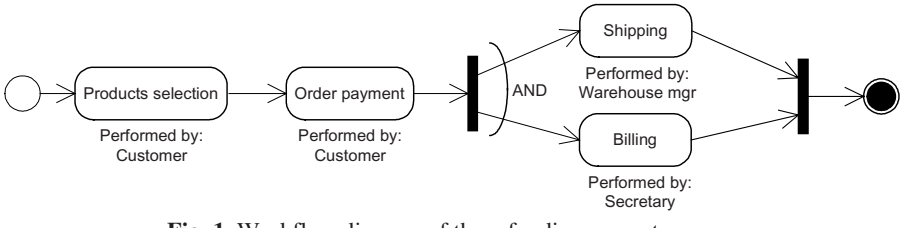

Fig. 1. Workflow diagram of the refunding request process

2.2 Hypertext Modeling

For hypertext modeling, we use the WebML notation[5, 6, 14], a conceptual language for specifying Web applications developed on top of database content described by a E-R diagram. A WebML schema consists of one or more *site views*, expressing the Web interfaces that allow the different user roles to browse or manipulate the data specified in the underlying E-R schema. A *site view* contains *pages*, possibly clustered in *areas*, typically representing independent sections of the site. Pages enclose *content units*, representing atomic pieces of information to be published (e.g., indexes listing items from which the user may select a particular object, details of a single object, entry forms, and so on); content units may have a *selector*, which is a predicate identifying the entity instances to be extracted from the underlying database and displayed by the unit. Pages and units can be connected through *links* of different types to express all possible navigation.

Besides content publishing, WebML allows specifying *operations*, like the filling of a shopping cart or the update of content. Basic data update operations are: the creation, modification and deletion of instances of an entity, or the creation and deletion of instances of a relationship. Operations do not display data and are placed outside of pages; user-defined operations can be specified (e.g., e-mail sending, e-payment, ...), and operation chains are allowed too.
Fig. 2 shows a simplified version of the two areas of the Customer site view of the e-commerce site example, whose workflow have been illustrated in Fig. 1: the *Products* area allows guests to browse products, by selecting in the *Home* page the product group from an index (*ProductGroups*). Once a group is selected, all the products of that group are shown in page *Products*. The *Mailing List Subscription* area allows the user to subscribe to a mailing list through a form. The submitted data are used to modify the profile of the User, by means of a modify operation called *Modify Subscr*, which updates the instance of entity *User* currently logged.

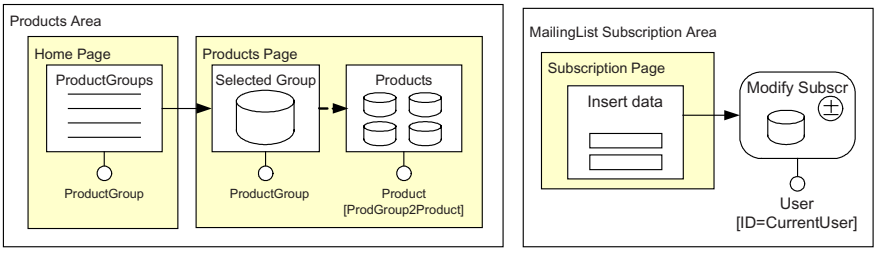

Fig. 2. WebML site view diagram featuring areas, pages, content units, and operations.

2.3 Extending Hypertext Modeling to Capture Processes

In the specification of a Web application supporting business processes[3], the data model, which is normally used to describe the domain objects, is extended with user-related and workflow-related data, and the hypertext model is enriched by a set of primitives enabling workflow dependent content of pages and navigation.
Process metadata. Data modeling is extended with the metadata used to represent the runtime evolution of processes as shown in Fig. 3. The schema includes entities representing the elements of a WfMC process model, and relationships expressing the semantic connections between the process elements.
Entity *Process* is associated with entity *ActivityType*, to represent the classes of activities that can be executed in a process. Entity *Case* denotes an instance of a process, whose status can be: initiated, active, or completed. Entity *ActivityInstance* denotes the occurrence of an activity, whose current status can be: inactive, active and completed. Entities *User* and *Group* represent the workflow actors, as individual users organized within groups (or roles). A user may belong to different groups, and one of these groups is set as his default group, to facilitate access control when the user logs in. Activities are "assigned to" user groups: this means that users of that group can perform the activity. Instead, concrete activity instances are "assigned to"

individual users, who actually perform them. If needed, the model can be enriched at will with new relationships to represent more complex assignment rules.

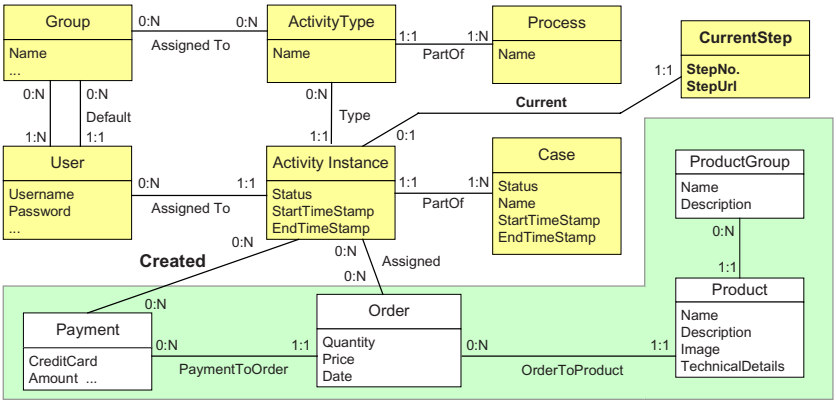

Fig. 3. Data model incorporating workflow concepts and exception handling information

Application data is described by a usual E-R model representing information involved in the current application. In our example, as depicted in the boxed part of Fig. 3, we model a catalog (in which each *Product* belongs to a *ProductGroup*), the *Orders* that the user submits and the *Payment* details. Orders are *assigned* to Activity Instances in which will be processed, whilst Payments are connected to the Activity Instances in which they are created. These relationships associate metadata concepts to application information. In general, the designer can specify an arbitrary number of relationships between the application data and the workflow data, which may be required to connect the activities to the data items they use. Note that minimum cardinality of these relationships is typically 0, since in most cases each activity instance is not associated to all the application data, but only to a very small set of objects.

This schema already includes metadata for supporting exception handling information. Such data are represented in bold face. In particular, the *Created* relationship and the *CurrentStep* entity are needed for supporting recovery policies for exceptions. Their use will be explained in the sequel of the paper.

Workflow hypertext primitives. In order to enact the process, some workflow-specialized hypertext primitives are also necessary to design interfaces capable of producing and consuming such metadata. At this purpose, a few additional primitives are introduced in WebML for updating process data as a result of activity execution, for accessing the data associated with a specific activity instance, and for expressing the assignment of data objects to an activity instance eventually to be executed.

The portion of hypertext devoted to the execution of an activity must be enclosed between the two workflow-related operations shown in Fig. 4(a): *start activity* and *end activity*. These operations are triggered respectively by incoming and outgoing links of the activity and have the side effect of updating the workflow data. Specifi-

cally, starting an activity implies creating an activity instance, recording the activity instance activation timestamp, connecting the activity instance to the current case (relationship PartOf), to the current user (relationship AssignedTo), and to the proper activity type, and setting the status of the activity instance to "active". Symmetrically, ending an activity implies setting the status to "completed" and recording the timestamp.

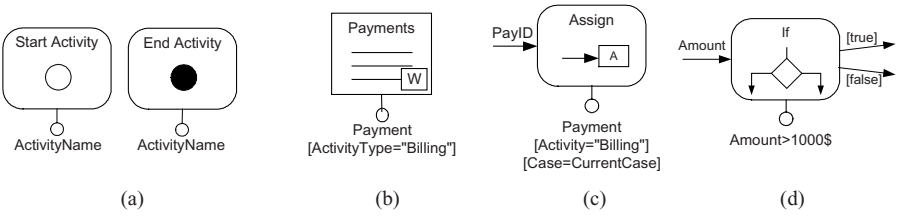

Fig. 4. *Start Activity* and *End Activity* operations (a); workflow-aware content unit notation(b); graphical notation of the *Assign* operation (c) and of the conditional operation (d)

The *Start Activity* operation can also be marked as the *starting case activity*, when the activity to start is the first one of the entire process; dually, the *End Activity* operation can be tagged as the *end of the case*, thus recording the general status of the process.
Workflow-aware content units can be used for retrieving the data objects related to a particular activity. These units are like the regular WebML content unit but are tagged with a "W" symbol denoting a simplified syntax for their selector, which shortens the expression of predicates involving both application data and workflow data. For example, Fig. 4 (b) shows a workflow-aware index unit that retrieves all the instances of entity *Payment* that have been assigned to an activity of type "Billing".
The *assign operation* is a WebML operation unit that connects application object(s) to an activity instance, for which an activity type, a case and possibly a user are specified. Fig. 4(c) shows the graphical representation of the assign operation, which assigns a *Payment* to the activity called "Billing" for the current process case.
The navigation of a hypertext may need to be conditioned by the status of activities, to reflect the constraints imposed by the workflow. Two dedicated operations called *if* (see Fig. 4(d)) and *switch* operations allow conditional navigation, performing the necessary status tests and deciding the destination of a navigable link.
Mapping rules have been defined from WfMC-based workflow description to WebML hypertexts enhanced with workflow primitives [3].

2.4 Fine Grained Description of Activities

To study in a simple and effective way the exception handling problem, we define some new concepts that describe the structure of activities.
We call *step* a hypertext page belonging to an activity. Steps are univocally numbered within an activity. Between two subsequent steps there can be a chain of operations, which is 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 (server-side failure is addressed by standard WebML mechanisms, like KO links [6]).
We define the *current step* 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 *CurrentStep* entity of the workflow metadata schema (Fig. 3). Within a process case, it is always possible to retrieve the currently active activities, and for each of them the current step.
The current step has 2 important properties: *(i)* it is always uniquely defined for an active activity; *(ii)* it gives us a correct idea of the progress of the activity.
It is important to notice that if the client uses the back and forward buttons of the browser, the current step of the activity does not change, since the client does not make any request to the server. Moreover, by clicking the *back* button we do not roll back the operations between consecutive steps, we just reload an old page.

3 Critical Situations and Exception

Within the execution of a process, exceptional situations can occur, due either to user behavior or to system failures. We define a critical situation as an incorrect browsing behavior of the user (*user-generated exceptions*) or a technical failure of the system (*system failures*).

3.1 User-Generated Exceptions

This section presents the critical situations that can arise from wrong browsing behavior. For Web context, this problem is much more relevant than for traditional applications. The most evident examples are *back* and *forward* buttons of a Web browser, that allow the user to explore the hypertext of the Web application in a free way, while a workflow scenario has usually a strictly forced execution/ navigation structure, and its steps must be executed in the proper order. Moreover, the user is able to jump without restrictions from an application to another. Back and forward buttons let the user go outside the pages of an activity still active or move back to a completed activity and try to resume its execution. With respect to workflow activities, improper browsing can be of three types:
(i) *improper inbound browsing:* the user gets into a workflow activity without executing the *Start activity* operation, for example by clicking repeatedly on the *back* browser button, until a previously executed activity is reached;
(ii) *improper outbound browsing:* the user, during the execution of an activity, follows a *wrong* navigational path, exiting the activity without passing by the *End activity* operation. In this case the user leaves the pages of the current activity, either by pressing repeatedly the back browser button or by following a landmark link (i.e., a link which is always clickable within the whole Web application). In this way, the user can potentially start an arbitrary number of activities, since he can try to start a

new activity beside the current one. Moreover, the user left an activity in status Active, which cannot proceed, and thus remains halted;
(iii) *improper internal browsing:* the user, during the execution of an activity, presses the back button of the browser one or more times reaching a previous page of the same activity, and then clicks on a link, trying to repeat part of the activity. In this way, the user is in a page that is different from the current step of the activity, since the page from which the user resumes the browsing is different from the last page requested to the server;
(iv) *wait:* the user does not request a page to the server for a given amount of time, after which a timeout expires and the user session ends up. A *Session End* exception is generated, and this behavior collapses in a system failure.

3.2 System Failures

System failures can occur both at client and at server side. *Client-side failures* are problems that are generated by system breakdown, that is either a client crash or a network failure. We do not consider server-side failures, since this problem for Web-based workflows can be addressed in the same way of traditional workflow systems, and several recovery theories and techniques already exist for this context (e.g., rules based on active rules [4]). System failures result in a Session end exception at server-side. To discover client-side failures, HTTP session is a standard technique employed in Web applications. After a session has been established, a network failure or a client failure will result either in the client not performing a request to the server for a given amount of time, or in the server being unable to send the response back to the client. When the server recognizes that the client is no more reachable, it will end up the user session: client-side failures can be captured at application level by generating a *Session End* exception. In this sense, client failure and network failure will be indistinguishable and will be collectively denoted also as *Crash* situations.
After a crash situation the activity instance executed by the user remains in Active status, but is not completed. This means that the activity execution cannot proceed, since the user lost his session, and if he tries to login he can only see the activities that are in Inactive status (ready to be executed). Typically he is not allowed to perform activities potentially in execution (i.e., in Active status).
If the activity instance is not recovered, the whole process case will possibly be stopped, if there are other activities waiting for the completion of the crashed one.
A thrown Session End exception will help to track the crash for later recovery

3.3 Inconsistencies

Data and process inconsistencies can arise from system failure and incorrect browsing behavior. Each of them will be addressed with a different approach:
1) *activity/process halt:* one or more activities (and the processes they belong to) get halted and cannot be resumed or concluded by the user. These problems are detected after they take place and are recovered by means of appropriate policies;

2) *inconsistent database:* one or more database tuples are created or destroyed in an unexpected way, resulting in an inconsistent database and workflow application; these problems are caused by incorrect browsing behavior, and will be handled in a preventive way, by detecting the user faults and generating an exception before they result in a failure.

4 Exceptions and Recovery Policies

As we have seen in previous sections, if a critical situation occurs, the workflow application might be in an inconsistent state due to the presence of a halted activity, i.e. an activity in status Active that cannot proceed. The need arises to recover the halted activity to bring the workflow application back to a correct state and let the process execution proceed. To address the problem, we define the concepts of exception and recovery policy.

4.1 Exceptions

To manage critical situations and to prevent/recover inconsistencies, we introduce the concept of exception. An *exception* is an event that is thrown by the system, as a consequence of a critical situation that is occurred.
An exception is either synchronous, if it is thrown after a page request, or asynchronous, if it is not tied to a page request but can occur independently. In case of synchronous exceptions, the user navigation can be immediately affected since the server can decide to provide the user with a different page depending on the caught exception. On the other hand, the only asynchronous exception that we will consider is Session End. It cannot influence immediately the user browsing, since he already disconnected from the application (his session is no more valid). Table 1 resumes the characteristics of exception types.

Table 1. Types and properties of the exceptions

Exception Type	Session Status	Addressed Problem
Asynchronous	Inactive	Technical Failure Incorrect Browsing Behavior
Synchronous	Active	Incorrect Browsing Behavior

Exceptions to be managed in order to guarantee the correctness of workflow-based Web applications are the following:
1) *Session End:* the user disconnected the client, or a failure happened on the network or at client-side. These events are undistinguishable from server side;

2) *Activity Already Active:* the user is trying to start an activity when there is another activity already active in his session;
3) *Wrong Starting Page:* inside an activity, an action has been performed in a page that is not the last one that the user has visited;
4) *Action By completed Activity:* an action has been performed within an activity that has been already closed.

In the following section we will discuss all possible critical situations and exceptions that can be generated.

4.2 Recovery Policies

We define a *recovery policy* (for a halted activity) as a collection of operations that we perform on the activity and on the related data in order to bring the workflow application to a correct state and to let the process proceed.
Policies can be classified with respect to three orthogonal dimensions:
- **policy direction**, that considers the way in which a coherent status of the process is reached: the policy can try to recover a correct status that was previously visited by the workflow application (*backward policy*), or can try to move to a new correct status that was not previously visited by the workflow application (*forward policy*).
- **policy definition**, that considers who defined the policy. In this sense, we can have policies defined either by the workflow design framework (*predefined policy*) or by the web designer (*user-defined policy*, also known as *compensation chain*).
- **policy execution**, that considers whether the policy is applied in an automated way (*automatic policy*) or in a manual way (*manual policy*). In the former case the policy is automatically applied by the workflow engine after an exception is caught and the engine detects a halted activity. In the latter case a user (the activity executor or another suitable user) can choose the policy to execute through a Web interface (*recovery page*), which is eventually reached after the activity interruption, through an explicit login of the user (Fig. 5).

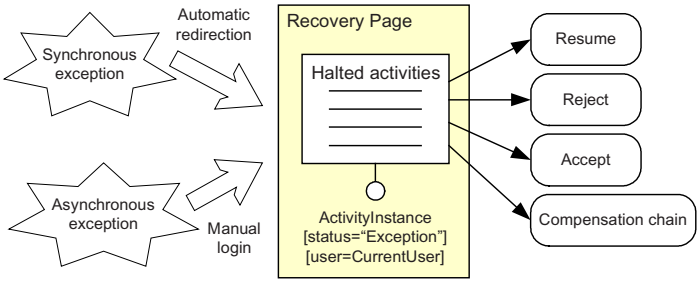

Fig. 5. Manual policies for synchronous and asynchronous exception management

Policies for Synchronous and Asynchronous Exceptions. Policy application can be affected by the type of the exception to be managed. In particular, we will apply different policies depending on the fact that exceptions are synchronous or not.

When a synchronous exception occurs the user session is still active. To take advantage of this fact we consider only manual policy for synchronous events: when the exception occurs the user will be redirected to a recovery page and will choose the most appropriate policy(either predefined or user-defined) for the halted activity.
When an asynchronous exception (i.e., a Session End) occurs, the user session is not connected any more and it is not possible to immediately apply manual policies. Therefore we consider both automatic and manual policy for asynchronous events.
Automatic policies are applied automatically and transparently to the user, while manual policies are applied by the user itself, when he starts a new session through a new login. At that point the user can go in a recovery page and choose the best policy to apply. This behavior is depicted in Fig. 5, together with the predefined policies that are described in next section.

4.3 Predefined Policies

Our framework offers three predefined policies: Accept, Reject and Resume. To better understand their behavior, we consider a very simple example, consisting in the order payment activity, as described in Fig. 6: the activity starts, a payment is created and connected to the order, and the user fills up a form with his credit card data. Then, the payment is performed (through a black-box service) and the payment status is updated. If an exception occurs the current step of the activity will be step 1 (since it's the only step of the activity).

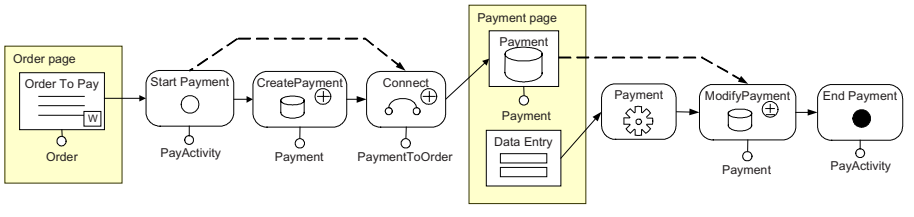

Fig. 6. Payment activity. There is just one step (the payment page), a preceding chain of operation (comprising the create payment unit and the connect unit) and a following chain (comprising the unit for the payment and the modify payment)

Accept policy. It accepts the operations already done by the halted activity, setting the activity status to Completed, executing all the data assignment and activating all the proper following activity. The process can proceed, but it may happen that part of the halted activity was not executed. The accept policy is a forward policy, since it tries to bring the workflow application to a correct status not previously visited, by simply assigning the status Completed to the halted activity.
This policy is suitable only for activities that have some non critical parts, that can be omitted. In all the other cases, it has resulted ineffective, since it leaves the activity results meaningless, thus damaging the whole process case execution. For example, suppose that an exception occurs in the payment activity in Fig. 6. Current step is 1, and if we apply an accept policy, we will consider the activity executed even if the

payment unit has not been performed. The process will be enabled to continue, even if the payment has not been performed. Therefore, in this case the accept policy is not a correct choice.

Reject policy. It deletes the data created by the activity, trying to recover the initial state of the database before the activity execution, and assigns the Inactive status to the activity. The reject policy is a backward policy, since it removes the data created by the activity (and all the relationships with connected objects), tries to recover the initial state of the database before the activity execution, and assigns the Inactive status to the activity. Reject policy is not a full rollback mechanism, since not all the operations executed by the activity are undone (i.e., deletion and modification results are kept as they are). Indeed, we don't want to implement a transactional system, with data versioning and so on. In this way, this policy can be implemented simply by means of a "*Created*" relationship, that connects the Activity Instance to the objects created in the activity itself (an example can be seen in Fig. 3). Once the reject is invoked, the activity is set to Inactive (ready to start) and all the *Created* objects are removed. Thus, reject is an approximate recovery of the initial state of the activity. This behavior partially limits the effectiveness of the policy but improves its efficiency, avoiding a performance burden resulting from a complete track of all the operations of the activity. Reject policy is suitable for all the activities that should be completely performed, and whose core task consist in creation of objects. With this policy, users can be asked to complete the activity repeatedly, until it is successfully finished. From empiric evaluations, this case results to be very frequent.

If we go back to the payment activity example (Fig. 6), by applying the reject policy in case of exception, we will delete the created instance of payment entity and the instance of the relationship PaymentToOrder, thus canceling the effect of both the create payment unit and the connect unit. The activity is ready to be restarted and the data are in a consistent status.

Resume policy. This policy lets the user resume browsing from the last visited page of the activity before the failure. This policy can be applied only by manual choice of the user, while the first two can be applied both automatically and manually. Browsing is resumed from the last page of the activity generated by the server, i.e. the current step. Note that operations with side effect are not improperly triggered by this policy: if the side effect occurs between the previous and the current page, it is not executed twice, because the user is provided with the url pointing directly to the page; if the side effect occurs after the current page, it has not been executed yet, otherwise the current page should point to the next one.

Resume is a backward policy, since it brings the application and the workflow to a correct state that was previously visited. Indeed, if an exception occurs, either the user session is expired or the page that is shown to the user is different from the current step. The user cannot proceed with the execution of the correct activity, and the whole process status is incorrect. As we said before, the resume policy can only be applied through manual intervention by the user. This can be achieved by providing to the user a recovery page, in which he can see the activities in incorrect status, and can decide to resume them. By reloading the last page generated by the server on the user browser, the activity execution can proceed (e.g., in Payment activity example in Fig.

6, the resume policy lets the user reload the payment page and complete the payment).

4.4 Compensation Chains

To allow a more fine-grained exception handling, we allow the designer to define his own recovery policies (e.g., sending warning emails to users, or implementing full rollback capabilities for specific activities). This solution will be adopted to manage the most critical activities only. The user can define operation chains that are triggered by exceptions. We provide a new unit (called CatchEvent unit), with the following parameters: ExceptionType, ActivityType, ActivityInstance. Exception type and Activity type are specified at design time and define the situation in which the compensation chain is triggered. Activity instance is a runtime parameter, whose value is available to operations of the chain, for retrieving further related data. Note that, since triggering and execution of compensation chains is completely automatic, no pages involving user interaction are allowed within chains. Fig. 7 shows a sketched example of compensation chain for the Payment activity depicted in Fig. 6.

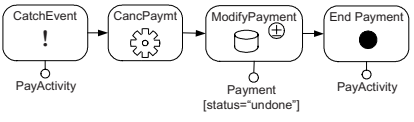

Fig. 7. Payment exception compensation chain. The payment is canceled and the information about it are update into the database

5 Implementation Experience

The concepts presented in this paper have been proved valid on the field, since a prototype implementation has been developed and used to design sample applications. The implementation extends a commercial tool called WebRatio[15], which allows to design and automatically generate Web applications from WebML models. Our extension provides the workflow metadata schema and all the units presented in Section 2.3. Moreover, new units for granting automatic policies enactment are available (Accept, Reject and Resume units).

Several case studies exploiting exception handling capabilities have been implemented, thus validating and refining the approach. The results of this research, which is part of the WebSI project, funded by the EC's 5th framework, has been used by the partners of the WebSI project for pilot applications, and by other projects.

Among them, Acer Business Portal (that includes remote service calls for providing location and driving information to users, and workflow-based interaction between Acer and its commercial partners), and MetalC project[11], which is the most complex among the application we have developed, since it includes a set of B2B portals

(one for each business partner). The purpose of the project is to allow business interactions between Italian companies of the mechanical field 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 critic. 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 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 is applied: 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.

6 Conclusions

In this paper we proposed a conceptual approach to exception handling within workflow-based Web applications, described through a metadata model and a set of primitives to be used into hypertext specification. To manage critical situations, we proposed an approach based on exception handling (some Java implementation already exists that could be used to support this approach [17]), and definition of predefined and user-defined policies, that have been tested on the field.
The main advantage of our approach stands in allowing to define exception handling and compensation chains without lowering the abstraction level of the design.
Future work will address refinement of the implementation, to allow a more seamless and transparent integration of exception handling within WebML specification, to avoid the need of explicitly specifying in WebML all the basic steps of exception handling. A second research direction is towards study of exception handling in remote service calls. Some preliminary considerations have been done, and we expect an approach similar to the one we have studied for workflow-based Web applications.

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. Brambilla, M., Ceri, S., Comai, S., Fraternali, P., Manolescu, I.: Specification and design of workflow-driven hypertexts, Journal of Web Engineering, Vol. 1, No.1 (2002).
4. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G., Specification and implementation of exceptions in workflow management systems. ACM Transactions on Database Systems, Sept. 1999, (Vol. 24, No. 3), pp. 405-451.
5. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. WWW9/Computer Networks 33(1-6): 137-157 (2000).
6. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications, Morgan-Kaufmann, December 2002.
7. Conallen, J.: Building Web Applications with UML. Addison Wesley (OTS), 2000.
8. Hagen, C., Alonso, G.: Exception Handling in Workflow Management Systems, IEEE Transactions on software engineering, October 2000 (Vol. 26, No. 10), pp. 943-958
9. IBM MQSeries Workflow Homepage: http://www.ibm.com/software/ts/mqseries/workflow/v332/
10. Oracle Workflow 11i: http://www.oracle.com/appsnet/technology/products/docs/workflow.html
11. MetalC project Homepage: http://www.metalc.it
12. Miller, J. A., Sheth, A. P., Kochut, K. J., Luo Z. W.: Recovery Issues in Web-Based Workflow, CAINE-99, Atlanta, Georgia (November 1999) pp. 101-105.
13. Schwabe, D., Rossi, G.: An Object Oriented Approach to Web Applications Design. TAPOS 4(4): (1998).
14. WebML Project Homepage: http://www.webml.org
15. WebRatio Homepage: http://www.webratio.com/
16. Workflow Management Coalition Homepage: http://www.wfmc.org
17. Ofbiz WF Java implementation: http://www.ofbiz.org/api/components/workflow/build/javadocs/