UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

# SUPPORTING EFFECTIVE UNEXPECTED EXCEPTION HANDLING IN WORKFLOW MANAGEMENT SYSTEMS WITHIN ORGANIZATIONAL CONTEXTS

Hernâni Raul Vergueiro Monteiro Cidade Mourão

DOUTORAMENTO EM INFORMÁTICA
(Engenharia Informática)

2007

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

# SUPPORTING EFFECTIVE UNEXPECTED EXCEPTION HANDLING IN WORKFLOW MANAGEMENT SYSTEMS WITHIN ORGANIZATIONAL CONTEXTS

Hernâni Raul Vergueiro Monteiro Cidade Mourão

DOUTORAMENTO EM INFORMÁTICA

(Engenharia Informática)

Tese orientada pelo Prof. Doutor Pedro Alexandre Mourão Antunes

2007

# Resumo

Os Sistemas de Gestão de Fluxos de Trabalho (SGFT) suportam a execução dos processos organizacionais. Os processos são modelados com recurso a linguagens de programação de alto nível que especificam a sequência de tarefas que a organização tem de realizar e os recursos necessários. No entanto, os processos organizacionais nem sempre têm um fluxo previsível que possibilite uma modelação ajustada a todas as situações que se encontram no dia-a-dia das organizações. Sempre que existe um desajuste entre o modelo e a realidade organizacional encontrada pelos utilizadores, estamos na presença de uma excepção. As excepções são eventos que se verificam com frequência e que obrigam as organizações a se suportarem por sistemas flexíveis que permitam o ajustamento às solicitações concretas que surgem no dia-a-dia. A flexibilidade deve ser complementada com robustez de forma a assegurar a fiabilidade do sistema mesmo em condições extremas. No nosso trabalho, introduzimos o conceito de resiliência dos SGFT que contempla estas duas características: robustez e flexibilidade.

O objectivo principal do nosso trabalho é aumentar a resiliência dos SGFT.

A primeira etapa da abordagem consistiu na caracterização dos eventos que requerem resiliência no sistema. A taxonomia mais adoptada distingue falhas de sistema, falhas de aplicações, excepções esperadas e excepções não esperadas. Tornámos esta taxonomia mais detalhada através da definição do contínuo entre excepções esperadas e não esperadas onde identificámos três classes: 1) excepções esperadas verdadeiras, quando o evento é equivalente a um esperado existindo procedimentos definidos na organização para o seu tratamento; 2) excepções esperadas semelhantes, quando o evento é semelhante a um esperado

embora não equivalente e os procedimentos existentes poderão ser aplicados com ajustes pontuais; e 3) excepções não esperadas efectivas, são eventos para os quais não existe conhecimento na organização que possa ser utilizado para o seu tratamento. O envolvimento dos utilizadores no tratamento do evento aumenta quando nos deslocamos das excepções esperadas para as excepções não esperadas uma vez que o conhecimento existente na organização sobre o evento diminui. Se não existe conhecimento na organização sobre o evento, o sistema não pode estar preparado para reagir de forma automática. Esta classificação foi ainda enriquecida com uma nova dimensão que distingue a capacidade da organização para definir um plano de reacção antes de iniciar as actividades de recuperação. As excepções para as quais é possível definir um plano de reacção são denominadas excepções com possibilidade de planeamento, enquanto as excepções para as quais não é possível definir um plano são denominadas ad hoc. A área das Ciências Empresariais relaciona a capacidade de planeamento com a incerteza associada à tarefa. Quanto maior a incerteza, menor a capacidade de planeamento da organização. Quando os operadores não conseguem definir um plano antes de iniciarem as actividades de recuperação, a situação tem de ser ultrapassada com recurso a com recurso a actividades de resolução de problemas – actividades não estruturadas. A solução que desenvolvemos destina-se a suportar os operadores no tratamento de *excepções não esperadas efectivas e ad hoc*.

As excepções foram abordadas segunda a perspectiva das Ciências Empresariais. Os factores de contingência que condicionam os subsistemas internos das organizações foram discutidos com o objectivo de perceber o impacto da solução na gestão das organizações.

Os sistemas existentes na literatura para aumentar a resiliência em SGFT foram agrupados segundo as suas características em cinco níveis: 1) abordagens sistémicas para o tratamento de falhas; 2) abordagens sistémicas para o tratamento

de excepções esperadas; 3) abordagens humanísticas com restrições e baseadas em intervenções pontuais; 4) abordagens humanísticas com restrições e baseadas em intervenções com suporte metamodelos; 5) abordagens humanísticas não restringidas para suportar actividades não estruturadas. A grande maioria dos sistemas existentes apenas suporta os primeiros quatro níveis. De todos os sistemas estudados, apenas um suporta o nível cinco. Concluímos, então, que existe uma área de investigação ainda não explorada.

O nível cinco é o mais exigente e requer intervenções sem restrições dos operadores na execução dos fluxos de trabalho. O sistema que propomos destina-se a suportar as intervenções sem restrições dos operadores – actividades não estruturadas. Dois requisitos do sistema foram definidos desde o início: 1) sistema completo; e 2) sistema aberto. O requisito do sistema completo especifica que os operadores podem efectuar as intervenções que entenderem sem qualquer restrição imposta pelo sistema. Os operadores devem poder efectuar o mesmo conjunto de operações que executariam se não tivessem o suporte do sistema. O requisito de sistema aberto deriva da necessidade de suportar as actividades não estruturadas com informação relevante e actualizada sobre o estado do sistema e da envolvente. O sistema deverá suportar os utilizadores com mecanismos de recolha de informação que lhes permita compreender a situação de forma a poderem tomar a decisão mais adequada.

As restantes características do sistema a desenvolver foram obtidas da actividade de resolução de problemas: 1) o tratamento do evento resulta de um esforço que envolve diversos operadores e onde é fundamental a participação dos actores chave; 2) os operadores devem ser suportados por mecanismos de suporte ao trabalho colaborativo nas suas funções de diagnóstico da situação excepcional e na tomada de decisão sobre as actividades de recuperação mais adequadas; 3) o sistema perdeu o controlo sobre a coordenação das diferentes acções de recuperação, passando os utilizadores a serem os responsáveis pela sua orquestração; 4) o diagnóstico não está completo na primeira abordagem devendo ser reajustado à medida que mais informação sobre o evento vai sendo recolhida; 5) mecanismos de suporte à decisão, desenvolvidos tendo em conta a realidade organizacional em causa, devem ser integrados na solução de forma a auxiliar os operadores na tomada de decisão sobre as acções de recuperação; 6) os utilizadores devem ter a possibilidade de, em qualquer altura, consultar o histórico do evento.

As funções elementares dos utilizadores no tratamento de excepções são: detecção; diagnóstico; monitorização; e recuperação. As funções de diagnóstico e de monitorização/recuperação são entrelaçadas, uma vez que o diagnóstico não está terminado na primeira abordagem e vai sendo refinado com a informação recolhida das acções de recuperação e de monitorização vão sendo implementadas. Foi proposto um modelo de referência para a solução proposta que resulta de uma extensão efectuada ao modelo apresentado pela organização Workflow Management Coalition. A arquitectura da solução é obtida do modelo de referência e identifica os componentes e as interfaces do sistema. A solução é implementada por um modelo de fluxo de trabalho que reflecte as características mencionadas. A solução foi implementada e funciona sobre a plataforma de código aberto OpenSymphony.

A avaliação da solução foi estabelecida em quatro etapas distintas: 1) visão; 2) validação da execuibilidade; 3) testes de campo; 4) utilização da solução pelas organizações. As reacções do sistema de controlo de tráfego aéreo nos Estados Unidos da América durante aos eventos catastróficos de 11/9/2001 funcionaram como uma fonte de inspiração conceptual para o desenho da solução. O esforço investido na implementação da solução reflecte a nossa preocupação com a validação da possibilidade de implementação da solução. O sistema desenvolvido permite concluir que a implementação é possível. A validação da abordagem apenas será possível com recurso a estudo de casos. Os estudos podem permitir validar a solução e estudar as realidades organizacionais onde esta se aplica. Finalmente, a prova final sobre a relevância da solução será obtida pela adopção que vier a encontrar junto das organizações.

**PALAVRAS-CHAVE:** sistemas de gestão de fluxos de trabalho; excepções não esperadas; suporte colaborativo; actividades organizacionais não estruturadas.

## Abstract

Workflow Management Systems (WfMS) support the execution of organizational processes within organizations. Processes are modelled using high level languages specifying the sequence of tasks the organization has to perform. However, organizational processes do not have always a smooth flow conforming to any possible designed model and exceptions to the rule happen often. Organizations require flexibility to react to situations not predicted in the model. The required flexibility should be complemented with robustness to guarantee system reliability even in extreme situations. In our work, we have introduced the concept of WfMS resilience that comprises these two facets: robustness and flexibility.

The main objective of our work is to increase resilience in WfMSs.

From the events demanding for WfMS resilience, we focused on ad hoc effective unexpected exceptions as those for which no previous knowledge exist is the organization to derive the handling procedure and no plan can be a priori established. These exceptions usually require human intervention and problem solving activities, since the concrete situation may not be entirely understood before humans start reacting to the event. After discussing existing approaches to increase WfMS resilience, we have identified five levels of conformity.

The fifth level, being the most demanding one, requires unrestricted humanistic interventions to workflow execution. In this thesis, we propose a system to support unrestricted users' interventions to the WfMS and we characterize the interventions as unstructured activities.

The system has two modes of operation: it usually works under model control and changes to unstructured activities support when an exception is detected. The exception handling activities are carried out until the system is placed back into a coherent mode, where work may proceed under model execution control.

**KEY-WORDS:** workflow management systems; unexpected exceptions; collaboration support; unstructured organizational activities.

## Acknowledgements

I will never be able to express my profound gratitude to my advisor, Professor Pedro Antunes. He was always there, when I needed him, and I needed him lots of times for long periods. His enlightenment and openness to discuss the different topics associated to this work were crucial on various moments.

I would also like to thank the people from APSS for the never ending support they gave me, in particular Professor Quaresma Dias, Dr. José Sacoto, Eng. António Ramos, Dr. Beatriz Mendes, Dr.. Jorge Figueiredo, Dr. Marco Caferra, Vânia Mendes, Manuel Henrique, Eng. Alberto Silva and Cristina Raposo.

I had one of my best experiences as a researcher in the Institute of Databases and Information Systems in Ulm's University. Friendship was in pair with professionalism and I will never be able to express my gratitude to all of them. I will never forget Prof. Peter Dadam, Prof. Manfred Reichert, Dr. Stefanie Rinderle, Christiane Köppl, Eva Mader, Alexander Kaiser, Hilmar Acker, Markus Kalb, Michael Bauer, Ralph Bobrik, Ulrich Kreher and Rudolf Seifert.

I am also grateful to my old colleagues and friends from Autoeuropa, Eng. Fausto Santos and Eng. António Pinto that supported me on investing the possibility of using the solution in the company as a case study.

I will never forget the first push I received from António Almeida to initiate this work. Also, the never ending support I received from Carla Rosário and Lurdes Cerqueira that never refused their help when I needed it.

My friends Emilio Vilar and Luis Moreira for the stolen time, I am deeply grateful. To my friends from IPS, Ana Paula Pereira, Lina Ferreira, Patricia Dias, David Simões, João Baía, José Rebelo, Pedro Cunha and Vítor Barbosa for the support and encouragement they have always gave me.

The last but not the least, to the people that pay the biggest price for this work, Hernâni João and Luisa Cordeiro. Hernâni João for my absence and Luísa for all the inconveniences that a work of this nature always brings. To my parents and family for my absence and lack of availability.

# Table of Contents

## List of figures

# List of tables

## Acronyms

ACID – Atomicity, Consistency, Isolation and Durability

API – Applications Programmer Interface

ATM – Advanced Transaction Models

BPM – Business Process Management

DBMS – Database Management Systems

ECA rule – Event Condition Action rule

ECH – Exception Handling Workflow

JSP – Java Server Pages

OS – OpenSymphony is the open source suite of components used in the solution

OSWF – OSWorkflow project within the OpenSymphony suite that implements a workflow engine

RDBMS – Relational Database Management System

SOA – Service Oriented Architecture

UI – User Interface

WfMC – Workflow Management Coalition

WfMS – Workflow Management System

XML – Extended Markup Language

# Chapter 1

# Introduction

It is commonly accepted that organizational procedures embed the knowledge required to achieve some desired organizational goals. These artefacts are used to guide the flow of work within the organization defining what is(are) the next task(s) to be executed, the required resources, the tools needed and the expected outcomes. Workflow Management Systems (WfMS) are computer systems developed to support organizational procedures. They are based on the stated premise that procedures are able to define the details of the work organizations have to carry out in order to achieve the desired objectives. Procedures are usually transcribed in the form of processes that embed the coordination logic while involved actors are responsible for implementing specific tasks. In a traditional WfMS, the process is defined using a language that the system is able to interpret. The flow of work is then scheduled among participants by following the rules stated in the process model. Since process definition is separated from its execution, the system is much more flexible than traditional information systems, and any change to the procedure may be easily accomplished. Using a WfMS, the organization should be released from the task of routing the process and all related information through the different tasks and affected actors. By placing a computer with the role of controlling the flow of work among actors, the traditional notion of procedure as an organizational artefact that supports users on their daily operations is transformed into a new and more rigid standard that all users must follow strictly, i.e., since it is implemented under a computational control, every organizational activity will have to conform with the process definition that plays now the role of a script.

This original development of WfMS was biased by a rationalistic approach that organizations follow their procedures on a rigid way in order to achieve their goals [Suchman, 1983]. However, organizations also require flexibility when performing their daily operations and processes do not necessarily contain all the required information to accomplish the work. This clash between the original objectives of WfMS and the concrete user and organizational requirements lead to a difficult acceptance of these systems by their target market during the nineties [van der Aalst and Berens, 2001; van der Aalst et al., 1999]. Therefore, some research effort was invested to overcome this limitation.

## 1.1 The limits of traditional WfMS

It has been shown by various ethnographic studies that the idealistic smooth flow of work described in process models is not always the case [Suchman, 1983; Bowers et al., 1995]. Suchman questions [1987] the premise that procedures (plans) are used by actors as guiding mechanisms (scripts) by realizing their limitations in accounting for all situations founded in concrete scenarios. Users are always conditioned by the peculiarities of the situations that are not completely reflected in the plan and thus ad hoc activities must be carried outside the plan. Suchman emphasises these observations by stating that every action is *situated* by the contingencies of the environment under which it is accomplished. In this perspective, plans are post hoc reconstructions of situated actions and filter the peculiarities that characterize them, because they are biased by a rationalist thinking. This radical approach has established a dichotomy between plans as scripts or plans as resources for situated actions. Plans are considered resources for situated actions because they play the weak role of a map that guides actors in the space of the available actions, like a map guides a climber informing on the available paths, the dangerous locations and so forth. A map does not in any sense determine the sequence of steps. However, Schmidt [1997] discusses this dichotomy using examples to show situations where plans play a stronger role. The basic idea behind Schmidt's work is not to question the validity of Suchman research but to establish limits on its applicability, i.e., there are situations where the available plans (even though post hoc reconstructions of situated actions) can be used as scripts releasing users from the coordination activities and supporting them on the sequence of activities required to achieve a desired goal. As we will see on the next chapter, research in Organizational Sciences also shows that procedures have different impact levels depending on the type of activity or organization. We may summarize this issue stating that plans play different roles according to the concrete scenario where the work is accomplished.

The two scenarios identified on the previous discussion are referred on this dissertation as *unstructured activities* when users are performing their activities using map guidance behaviour, and *structured activities* when procedures play the stronger role of a script determining user actions. They should both be taken into account when designing systems to support organizational activities. However, WfMS are traditionally algorithm-based and developed with a special focus on supporting structured activities. One of the main disadvantages using these systems has been their lack of flexibility to adjust to concrete user demand, i.e., their inadequacy supporting unstructured activities [Abbott and Sarin, 1994; Blumenthal and Nutt, 1995]. An *exception* is therefore a scenario where the system is not able to support the users performing the required actions to achieve the organizational goals. From the above discussion it can also be said that when the plan (regarded as a resource) is not able to guide actors through the tasks, i.e., organizations face the applicability limits of the plan in a concrete situation, it can be said that we are in the presence of an exception, where the situated characteristics of actions should prevail over the prescribed ones. This discussion enables us to express the main problem addressed by this thesis:

*Difficulty adjusting WfMSs to real world scenarios.*

The same problem has been addressed by various researchers in the field. However, different solutions have been proposed by the research teams because there is not a clear understanding on the type of support user's demand on those situations. We believe the majority of the proposed solutions are biased by the rationalistic approach to the problem, where more primitives are inserted on the WfMS that is always under any sort of an algorithm-based control. Even when primitives are inserted to increase adaptability, they have their roots on the original model and therefore do not allow the latitude operators require. Therefore, the problem to be addressed is the difficulty that traditional WfMSs have coping with unstructured activities. We assume there will always be situations where users should be able to decide on what are the most suited activities to fulfil organizational goals without any kind of restriction imposed by the system based on a prescribed procedure. This statement imposes that on some situations users should have the flexibility they have at their disposal when they are not working with the support of a WfMS, i.e., all the flexibility they possess when working at their office with all the tools they can use to accomplish work. This is the *completeness requirement*, a core concept of our system that we describe in detail on Section 2.5.

## 1.2 The proposed solution

*The solution developed in this thesis is designed to support both behaviours: structured and unstructured activities.*

It is assumed that actors mainly adopt structured activities and thus work under model guidance. When facing an exception, the system is able to change to unstructured activity support. As mentioned in the previous section, during unstructured activities users should be able to implement any activity they desire without any restriction. The main idea behind this assumption is to increase the latitude of interventions available to users.

We should informally introduce the model consistency concept (cf. Section 2.1.1 for a formally definition) since it is the basis for some of the discussions that follow in this section. A workflow model is consistent if it allows a proper completion in all situations, i.e., when users are performing structured activities prescribed by the model they will allays reach the end and no tasks in middle are left to be executed.

A direct impact of the increased latitude of the interventions mentioned above is on the workflow model consistency. Contrary to existing systems (cf. Section 3.2), our system enables users to implement any activity they desire even if they insert inconsistencies into the workflow model. It is the user’s responsibility to decide if the activity should be implemented.

Figure 1.1 shows the state diagram of the proposed solution that governs the above mentioned behaviour for structured and unstructured activities support. The solution implements an exception handling service that initiates when an exception is detected and supports unstructured activities. As mentioned before, unstructured activities are map guided which means that user's actions should be driven by information collected from the system and environment to support decision making. On the other hand, these actions can involve more than one person from the same department or from different departments within the organization. Whenever more than one person is involved, some orchestration mechanism must be provided to assure coordination between the efforts of the various actors.

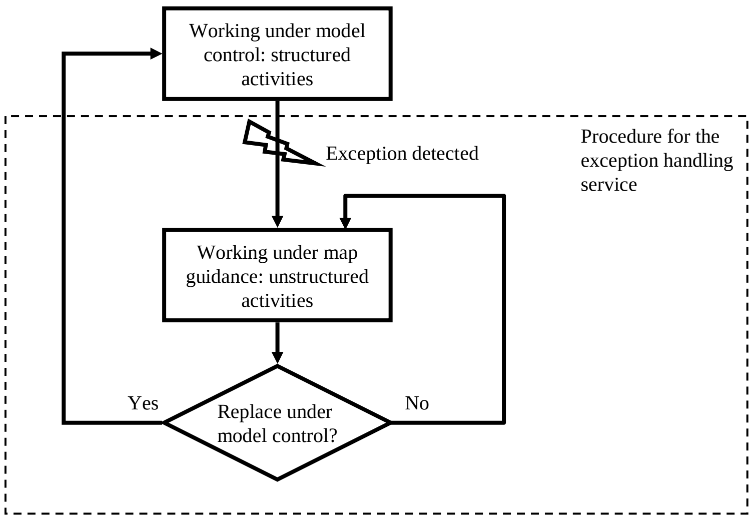

Working under model control: structured activities
Exception detected
Procedure for the exception handling service
Working under map guidance: unstructured activities
Yes
Replace under model control?
No

Figure 1.1. State diagram of the developed system to support structured and unstructured activities in a WfMS

Unstructured activities proceed until the system is driven back into a coherent state and can be placed back into model control. This is the last decision shown in the state diagram. This decision should take into consideration any inconsistency inserted into the model during unstructured activities support. As mentioned before, users should be able to implement the most suitable action during unstructured activities support without any system restriction – even if it requires inserting inconsistencies in the model. Therefore, all the inconsistencies should be removed before placing the system under model control. This activity can be implemented by the users during unstructured activities and may be supported by existing techniques that detect model inconsistencies (cf. Section 3.4). The process of implementing unstructured activities to react to an exception and bring the system back into a coherent state is the procedure for the *exception handling service*.

Whenever we mention the *proposed solution* in this dissertation we refer the computer based system that supports organizational activities. It is composed by the WfMS standard system that supports structured activities and by our developed functionality to support unstructured activities. Users are one of the entities within the overall organizational system that interfaces with the proposed solution. The boundaries and interfaces of the proposed solution are identified in Section 5.1.

Since supporting structured activities is a standard feature of WfMS, we will only discuss in detail the unstructured activities support. Therefore, we will pursue the main problem identified above in Section 1.1 by focusing on:

*Introducing unstructured activities features in a WfMS.*

We will describe the general characteristics of these activities and identify the main functionality the system should implement. It should be emphasized that the proposed solution aims to be generic and applicable to any application environment addressing the support of automated and human tasks in organizational environments, such as Business Process Management (BPM) and Service Oriented Architectures (SOA). In fact, the concrete implementation for the system is not a major concern to our approach. This thesis proposes a conceptual approach to manage business processes adjusting them to concrete organizational scenarios. The underlying system used to implement the system is not our research topic. Therefore, the recent research trends of BPM and SOA do not in any way collide or invalidate our approach.

We are also not concerned on studying a particular implementation scenario. However, in some specific applications there could be functionality that are of particular importance and should be provided. A decision support tool, able to process application specific data, is an important functionality when users have to make decisions on the recovery actions that will bring the system back into a coherent state. We recognize this facet by integrating a general tool that is adjusted to the concrete implementation.

The functionality that a system should implement to support unstructured activities identified in this thesis consider:

(i) Escalation;

(ii) Monitoring;

(iii) Diagnosis;

(iv) Communication;

(v) Collaboration;

(vi) Recovery;

(vii) Coordination;

(viii) Tools to determine the best solution;

(ix) History log.

An *escalation* mechanism is implemented to allow the involvement of several users in the exception handling effort. This escalation mechanism is used by the users affected by the exceptions to escalate the exceptional event to the upper levels of the hierarchy.

In contrast with structured activities where users have a procedure that prescribes all the required steps to achieve an organizational goal, during unstructured activities they have to decide the most suitable actions. Therefore, unstructured activities should be fed with updated and relevant information in order to understand the peculiarities of the exceptional situation at hand and to improve the decision making process on the most suitable activities to carry on. It is important to realize that these activities can, in some situations, be characterized as problem solving, where users do not know all the details about the situation and some information may have to be gathered to improve the diagnosis. Therefore, *monitoring* information about the system and the environment should be continuously collected and distributed to the involved actors. Even further, monitoring the effects on the system and on the environment of the implemented actions decided by the involved users to react to the situation is also important to understand their impact. This monitoring capability is the first functionality that should be implemented by the system.

This linkage to the environment to collect relevant information is one facet of the *openness requirement*. This requirement, that will be completed bellow, is another key aspect of the proposed solution.

Situation *diagnosis* is another important functionality to support user understanding the exceptional situation. The solution implements a situation description component used to classify the event according an established taxonomy. This classification may evolve over time as users change their perception of the event.

On the other hand, some exceptional situations will involve more than one user. In those situations it is critical to implement *communication* and *collaboration* mechanisms among them in order to exchange information that facilitates the creation of shared contexts and improve the common awareness of the situation. In a different dimension, the collaboration between involved users should be facilitated to support the decision making process. Communication and collaboration support is therefore another important functionality of the implemented solution. Even though the system implements communication and collaboration mechanisms, the users may use alternative methods (e.g., meetings or telephone conversations). Another facet of applying the openness requirement to the developed system is the capability to collect relevant contextual information about the usage of these external mechanisms and tools.

The solution must also enable users to intervene on the system after they decide the most adequate actions to handle the exceptional situation. The *recovery* mechanism enables these interventions. It is important to realize that during unstructured activities support the *coordination* facet of traditional WfMS must be relaxed since users decide on the most adequate activities to implement. Therefore, during the solution development, this facet was investigated and we propose a solution that manipulates the relevant coordination aspects for system and users.

The functionality *tools to determine the best solution* stands for some environments where the associated complexity is high and tools may be available to support decision making and situation diagnosis (e.g., in the already mentioned example of a lot manufacturing facility, the batches produced to respond to customers requests may be calculated using operations research tools). This functionality is oriented towards investigating decision making tools in the exception handling process.

Finally, the functionality *history log* stores old values of the situation description and the implemented activities. The log may be consulted during the event or on future similar events.

## 1.3 Main contributions

The first contribution of this thesis is a **proposal for a new reference model** for WfMS that is able to handle all type of exceptions. The reference model extends the WFMC’s reference model with a new component. The interfaces of the inserted component with the existing components were identified. The functionality of the component and the information flow on the interfaces were also defined.

Then, another important contribution is the **exception handling procedure and its integration in the WfMS**, according to the state diagram mentioned in the previous section. The identified states and the conditions under which exception handling takes place are defined, because they are key aspects to understand the proposed solution.

Supporting unstructured activities in organizational workflow is a research field that has not yet received the deserved attention by the researchers in the field. Therefore, another important contribution of the present dissertation is the **study of functionality, requirements, and conditions associated to the exception handling process**.

Another important contribution is the **identification of the situations that the solution is able to handle**. The related literature uses a broadly accepted classification that distinguishes expected from unexpected exceptions [Eder and Liebhart, 1995; Casati, 1998]. Expected exceptions are exceptions that can be predicted during the designed phase while unexpected exceptions can not be predicted and must be handled during runtime (cf. Section 2.3.1). However we have questioned this dichotomy by realizing that in some situations the organization may find similarities with some events that happened before. The organization can therefore use its past experience to derive advised behaviour on the present situation. Since the degree of similarity with previous events may vary, we propose a continuum from expected to unexpected exceptions. This continuum characterizes a set of the situations where the solution is applicable. The exceptions close to the unexpected limit of the spectrum are those without any previous organizational knowledge and where unstructured activities should prevail over prescribed ones. As a result of this discussion **we have proposed a new taxonomy** based on the expected-unexpected exceptions continuum. Another dimension was added to this taxonomy to characterize whether users may plan the complete reaction before any recovery mechanism is implemented. These two dimensions enabled us to identify the exceptions that should be handled with the support of an unstructured activity support service – ad hoc effective unexpected exceptions, are situations for which no knowledge exists in the organization that can be of any usage and that no a priori plan can be drawn.

Other important contribution is the **characterization of the conditions under which forward and backward jumps can be done in the model**. This kind of tests is particularly useful when the user wants to perform jumps in the model without compromising its consistency. However, different from the majority of the systems found in the related literature, we allow the user to jump even when inconsistencies are inserted. The user is advised and may proceed if desired. It is the user’s responsibility to choose the most adequate actions. As mentioned in the previous section, tests on model consistency are performed before the system is placed back under model control.

Finally, one fundamental contribution is the **implementation of the proposed solution in a real world workflow system**. The solution was implemented in the OpenSymphony (OS) [OpenSymphony, 2007] project, developed in the Java programming language and available to the open source community. The exception handling procedure is implemented by a dedicated model that runs on the same WfMS system as the organizational models. This implementation highlighted several issues related with exception handling that are of the most importance to implementers and to software engineering community.

## 1.4 The research context

The topic under investigation is still an hot topic deserving attention by the researchers in the field as the recent publications from Combi et al. [2006], Adams et al. [2005] with a recent PhD thesis by Adams [2007], Russel et al. [2006] and Vojevodina [2005] seem to fundament.

In our work, we purpose a new approach to unexpected exception handling that strongly differs from the majority encountered. During our work, we felt the necessity to validate the approach in real world scenarios.

However, finding effective examples of unexpected exceptions is difficult because if the exception is defined beforehand then it must be biased. An effective unexpected exception must always be brought from real life with proper documentation about the adopted strategies to handle the situation. Considering this limitation, two motivating examples are used to illustrate the solution: 1) a media report on the 9/11 catastrophic event, as experienced by the USA's air traffic control centre; and 2) a non-catastrophic but also unexpected event, where a WfMS must handle for the first time a client that went bankrupt in a real world organization. While this second situation is much less inspiring that the first one, it was indeed experienced by us during the implementation of a space rental management system for a Port Authority.

The 9/11 example will be used throughout this thesis as an inspiring event to establish our solution. Whenever we use the example, the text is inserted in a box to differentiate it from main text and facilitate the reading.

The 9/11 event was fundamentally selected because very rich information about the adopted exception handling procedures is available to the public [USA TODAY, 2007][1]. The overwhelming impact in society and strong political implications of this unique event were not within the selection criteria and are out of the scope of this research. On the other hand, as discussed in the previous section, an effective unexpected exception is an event for which the organization has no prior knowledge about the resolution. Therefore, this is a good example to motivate the discussion on how WfMS users react to this type of situations.

Considering regular air traffic control, every plane is a process instance and every route is modelled since the plain first checks in the air traffic control on the departing airport and until it checks out on the arriving airport. For instance, AA flight 11 route on 9/11 started at Boston and its model considered driving it to Los Angeles. At approximately 8:15 AM on that day, the air traffic control centre in Boston stopped receiving feedback from the airplane pilots and lost the transponder signal. Controllers also reported hearing a man with a strange accent in the cockpit. This combination of events originated an exception. Along with the description of our exception handling solution, facts from this real event will be used to exemplify how the proposed solution could be used to support exception handling.

One of the key decisions taken during this exceptional situation was to land every plane that was flying in the USA and Canada air spaces. According to FAA officials, facing this exceptional event, they "[…] decided not to write a new set of procedures for clearing the skies. They started to but scrapped the idea. They concluded that the FAA was better off relying on the judgment of its controllers and managers." From our perspective, this means that under such extreme conditions procedural control was considered worse than giving people access to the relevant updated information and letting them decide the best reactions to the concrete situation, i.e., map guidance was clearly favoured against model guidance.

Another important implication to our research can be drawn out from this quote from USA Today: "landing nearly 4,500 planes was a massive undertaking and a historic achievement. It required intense cooperation, swift decision-making and the unflinching work of thousands of people. Across the nation, controllers searched for alternate airports to land large jets." The mentions to intensive cooperation and swift decision making are crucial to our exception handling approach.

[1] the report was issued by USA Today based on interviews to more than 100 people involved in key decisions and data collected from other sources, such as FAA radar, air traffic control databases and a special software to analyze plane rerouting.

The second example was chosen because it results from a workflow implementation in a Port Authority organization within the work developed for this thesis. During the implementation, we had the chance to follow a real exceptional event since it was detected until the handling finished. The example helped us understanding the concrete user’s needs on these situations and how the system can support them. Several data related with this event was collected and the system users were interviewed to identify the adopted handling procedures and their relationships with the WfMS.

During the development of this work, the author had the opportunity to work for six weeks in the Institute for Databases and Information Systems in the University of Ulm with one of the most prominent research groups in the area of adaptable workflow management systems. The exchange of ideas enriched the concepts developed and the proposed solution.

Before starting this research work, the author has worked for four years in a car manufacturing company. This experience has made him aware of the organizational conditions under which work formalization and flexibility must coexist. The perspective obtained has helped on establishing the concepts developed in the present work.

## 1.5 Publications

The following papers containing partial results of the work developed during this thesis were published.

Mourão, H. and Antunes, P., (2003) Workflow Recovery Framework for Exception Handling: Involving the User, Groupware: Design,

Implementation, and Use., J. Favela and D. Decouchant. Lecture Notes in Computer Science, vol. 2806, pp. 159-167. Heidelberg, Springer-Verlag

Mourão, H. and Antunes, P. (2003) Supporting Direct User Interventions in Exception Handling in Workflow Management Systems, Workshop de Sistemas de Informação Multimédia, Cooperativos e Distribuídos – COOPMEDIA 2003, Porto, Portugal

Mourão, H. and Antunes, P. (2003) Suporte à Intervenção de Operadores no Tratamento de Excepções em Fluxos de Trabalho, 4ª Conferência da Associação Portuguesa de Sistemas de Informação, Porto, Portugal

Mourão, H. and P. Antunes (2004) Exception Handling through a Workflow. On the Move to Meaningful Internet Systems 2004: Coopis, Doa, and Odbase: OTM Confederated International Conferences, Coopis, Doa, and Odbase. R. Meersman and Z. Tari. Lecture Notes in Computer Science, vol. 3290, pp. 37-54. Heidelberg, Springer-Verlag.

Mourão, H. and P. Antunes (2005) A Collaborative Framework for Unexpected Exception Handling. Groupware: Design, Implementation, and Use. H. Fuks, S. Lukosch and A. Salgado. Lecture Notes in Computer Science, vol. 3706, pp. 168-183. Heidelberg, Springer-Verlag.

Mourão, H. and P. Antunes (2005) "Supporting Direct User Interventions in Exception Handling in Workflow Management Systems." Sistemas de Informação, 17, pp. 39-51. ISSN: 0872-7031.

Mourão, H. and Antunes, P. (2007) Supporting Effective Unexpected Exceptions Handling in Workflow Management Systems, Proceedings of the 22nd Annual ACM Symposium on Applied Computing, pgs 1242 – 1249, Seoul, South Korea,  ACM Press, ISBN 1-59593-480-4

The relevance for current research in the WfMS area of the published work may be attested by the citations that it has already deserved. Appendix D provides a list of papers that cited our work.

## 1.6 Organization of this thesis

We start in Chapter 2 by discussing the adjustment of WfMS to organizations. After revising the main concepts associated to WfMSs in Section 2.1.1, we propose a new reference model for a WfMS that is able to handle all types of exceptions in Section 2.1.2. Then, in Section 2.2 we revise the Organizational Sciences perspective on procedures and how they can be used to organize the work. The notion of exceptions as seen from this perspective is also discussed and the theory is also used to fundament our proposed solution. The chapter proceeds describing two perspectives that exist in the literature to classify exceptions in Section 2.3: a system and an organizational. The taxonomies are compared and merged to establish the taxonomy used in this thesis. In Section 2.4 we discuss and refine one classifying dimension that is common to both perspectives and we insert a new dimension used to identify the events requiring unstructured activities. Finally, in Section 2.5 two mandatory requirements to effectively support unstructured activities are identified: openness and completeness.

Chapter 3 starts by defining the resilient property of a WfMS. Resiliency concerns being robust to react to failures and flexible to handle unexpected exceptions. We then review the existing systems to increase resilience that were grouped into systemic and humanistic approaches. Systemic approaches, described in Section 3.1, are mainly designed to keep control in the system and handle failures and expected exceptions. They mainly increase robustness and have minor impact on flexibility. Humanistic approaches, described in Section 3.2., are designed to increase flexibility to handle unexpected exceptions. They support users adjusting business models to react to exceptions and migrating running processes to the new model. However, these approaches restrict the allowed interventions to maintain model consistency. In Section 3.3 we compare the reviewed approaches according to their impact on resilience and we identify five different consistency levels of support. We were also able to realize that few systems integrate the consistency required to support unstructured activities. To conclude the chapter, Section 3.4 discusses the existing modelling formalisms according to their expressiveness capability and to their impact on workflow changes. We also justify the adopted modelling formalism for this thesis and identify how changes may be implemented in the adopted formalism.

In Chapter 4, we establish the conceptual model of our approach. In the conceptual model we have completed the WFMC’s reference model introduced in Chapter 2, and described the concepts of organizational trajectory for the exceptional handling procedure, and organizational escalation of the procedure in the organizational hierarchy when users are involved in the handling process. We proceed with the state diagram of the solution that supports all the five consistency levels defined in the previous chapter. We settled the focus of this thesis on level 5, a system to support unstructured activities. Then, the basic functions required to support a level 5 system are described: detection, diagnosis, recovery and monitoring. In our approach, we advocate an intertwined play between diagnosis and the handling activities of monitoring and recovery, since it is considered that the diagnosis may not be complete at the first approach. This chapter concludes with classifications to describe the event and the handling strategies.

Chapter 5 is dedicated to the solution’s architecture and implementation. The solution’s architecture is derived from our extended reference model and the main components and interfaces with the WfMS and with the environment are identified. The solution is implemented by a dedicated workflow that implements the solution state diagram and the basic functions identified in the previous chapter. When an exception is detected, the exception handling workflow is instantiated initiating unstructured activities support. The chapter proceeds with the description of the constructs that automatic detect exceptions and with the recovery and monitoring operations users implement to bring the system back into a coherent state. The remaining part of the chapter is dedicated to the implementation details using the OS suite. Hence, we start by describing the suite and then how exception detection and recovery is implemented using the suite. Next, we explain the user interface with the service by using interface examples and we finish by describing the data model used to store exception related information.

In Chapter 6 we use the Port Authority and the 9/11 examples to validate our approach. The Port Authority example results from a real unexpected exception that we were able to follow whereas in the 9/11 we discuss how the solution could have been used by the air traffic control system. Finally, chapter 7 is dedicated to the conclusions and future work.

# Chapter 2

# WfMS in Organizations

The work processes carried out by organizations in their daily operations have been identified to belong to a continuum ranging from totally unstructured to completely structured [Sheth et al., 1996]. It is interesting to note that the majority of the available organizational information systems tend to fall close to both sides of the spectrum boundaries [Sheth et al., 1996], thus leaving a significant gap in between. Unfortunately, traditional WfMS fall into the highly structured boundary and thus contribute to this gap. WfMS emphasize the execution of work models, play the role of scripts in formal organizational structures and thus have a normative engagement [Schmidt, 1997]. Closer to the other end of the spectrum limits, Suchman [1987] proposes the notion of maps, which position and guide actors in a space of available actions, providing environmental information necessary to decision making but avoiding the normative trait. Email systems, the newly developed collaborative Web platforms sharing information among users that can also collaboratively change the content, and group decision support systems that can be found in meeting rooms are examples of systems that fall close to the unstructured limits of the spectrum. These systems do not have any model to be followed since no model is appropriate to support the interactions.

Since traditional WfMS fall close to the structured limits of the spectrum, they are inadequate to cope with unstructured processes that emerge in organizations. To support the continuum of organizational needs, WfMS should cope with the whole spectrum of structured and unstructured activities. This requirement has been identified by Ellis and Nutt [1993], when they realized that WfMS must be flexible to succeed. Also, Abbot and Sarin [1994], based on empirical evidence, claim it is necessary to integrate procedural and nonprocedural work in WfMS to react to exceptions. They define nonprocedural work as “unchoreographed interactions between people”. In our solution, we propose a system that is able to switch its behaviour from structured activity where operators are supported by model guidance to unstructured activity, characterized by map guiding operators, and then back to structure whenever required. In the WfMS community, nonprocedural work has also been designated *exception handling*, encompassing the set of actions aiming to react to a kind of event that is out of the scope of the work model [Abbott and Sarin, 1994].

The main objective of this chapter is to characterize the types of events that the solution developed in this thesis is designed to handle. It starts by establishing WfMS core concepts in Section 2.1. Section 2.2 discusses procedures and their usage inside organizations in the light of the Organizational Sciences perspective. The role of exceptions in organizations is also studied and the strategies to handle them according to management sciences are discussed. This perspective is also used to discuss the adequacy of the proposed solution. Section 2.3 introduces two taxonomies for exceptions based on two different perspectives: system and organizational. By identifying different exception dimensions and their impact on organizations the exception types can then be delineated. Even further, from the evaluation of the organizational impact important guiding behaviour during exception handling can be drawn. In Section 2.4, the exception taxonomies are used to identify the exception type handled by the proposed solution: ad hoc effective unexpected exceptions. Finally, in Section 2.5 we introduce the openness and completeness requirements.

## 2.1 Workflow management systems

The WfMS technology has evolved since its concepts were firstly outlined in the mid 70s [Nutt, 1996]. It has its origins on Office Information Systems as researchers realized that changes needed in the information systems developed to support clerical work were very difficult and expensive, because business logic was embedded in the system [van der Aalst and van Hee, 2002]. It usually involved the modification of complicated programs developed by different programmers and was time consuming and expensive. Researchers proposed to decouple the control flow from the elementary tasks that were required to fulfil the organizational processes. The control flow would then be specified in high level programming languages (including visual programming), while different computer applications implemented the tasks. The idea was to increase the system’s flexibility and adapting changing requirements or small model deviations. These changes could be accomplished trough changes in the high level language that could even be accomplished by the end user.

Some high level modelling languages were then proposed to decouple the process specification from its execution [Ellis and Nutt, 1980; Hammer et al., 1977]. Aalst and Hee [2002] mention this separation as one of the most important characteristics of a WfMS, because it enables the separation of the business process logic from the system. Any change in the business process can be implemented by a change made to the model that should be easily achieved and even be performed by the operators lacking programming experience. The WfMS concept was established as an automated system to support the execution of business processes within organizations.

As the field developed, some definitions were issued. In particular, Sheth et al. [1996] present a complete definition where the various facets of a WfMS are enhanced. The definition starts with business process: a collection of activities tied together by a set of precedence relations and having a common organizational objective. This involves distributing, scheduling, controlling and coordinating work activities among humans and information systems resources. This definition also embraces the organizational perspective of business processes, as a collection of tasks with the aim of achieving a common goal, and the system perspective, where the tasks must be coordinated, distributed, scheduled and controlled.

Sheth et al. define workflow management as the automated coordination, control and communication of work task, both of people and computers, as it is required to carry out business processes. This is performed by a workflow enactment service, which is controlled by a computerised representation of the organizational processes and provides the required services on a computer network.

The Workflow Management Coalition (WFMC) [WfMC, 2007] is a global organization where the main companies developing WfMS products are represented. It is among its objectives to develop standards in the field that facilitate interoperability among workflow systems developed by different vendors. Since the terminology defined by the coalition has encountered some acceptance in the area, the next section presents the WFMC concepts that are relevant for the present dissertation. The reference model developed by the coalition is also briefly described. This section is also used to define some core concepts used throughout this work. Then, in Section 2.1.2, the coalition's reference model is completed and a new model able to support unstructured activities according to the solution developed in this work is described.

### 2.1.1. WfMS components and main concepts

The WFMC definition for a WfMS is [WfMC, 1999] "a system that defines, creates and manages the execution of workflows trough the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications." Therefore, the WfMS interprets a process definition and creates, manages and defines the execution of workflows.

The definition is complemented with the WFMC reference model shown in Figure 2.1 [Hollingswoorth, 1995] that represents a generic WfMS with the five areas of functionality. Workflow Client Applications and Invoked Applications belong to the same area of executing tasks. For the sake of simplicity, we will refer to the areas of functionality, excepting executing tasks, as components of the WfMS system. The main purpose of the reference model is to specify the interfaces between components to assure interoperability. Process Definition Tools enable the design of process models while the Workflow Enactment Service interprets the process models and controls the instantiation of the processes and sequencing of activities. Workflow Client Applications and Invoked Applications represent the applications that implement the tasks and are invoked by users or by the workflow respectively. WfMS usually run on different systems and platforms being often classified as heterogeneous, autonomous and distributed [Worah and

Sheth, 1997]. The interface with other workflow enactment services is also represented in the figure.

According to the WFMC terminology and glossary [WfMC, 1999], workflow monitoring "is the ability to track and report on workflow events during workflow execution" and administration can include functions like initiating or suspending the execution of some task or model, reassign work items and control the versions of process definitions. The Administration and Monitoring component implements these features.

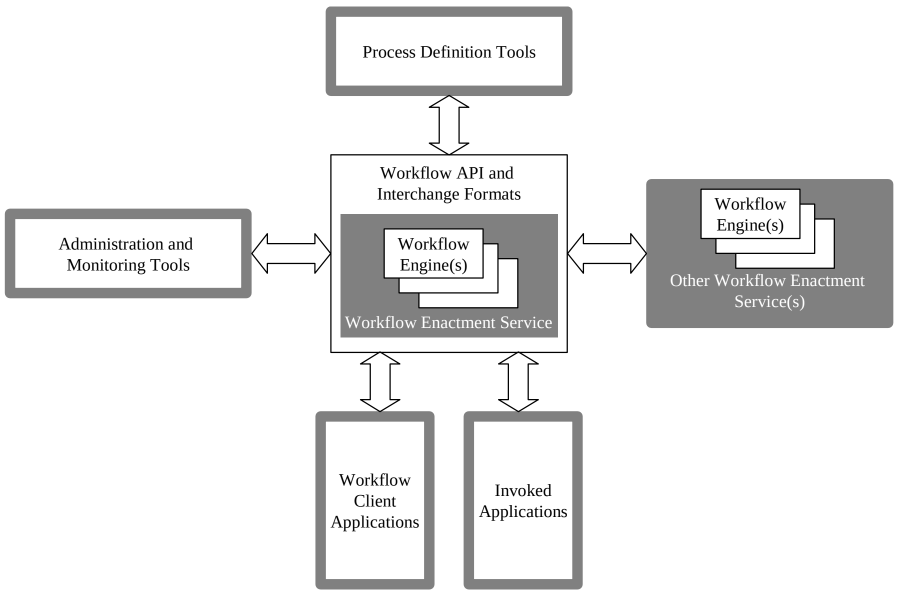

Process Definition Tools
Workflow API and Interchange Formats
Workflow Engine(s)
Workflow Enactment Service
Administration and Monitoring Tools
Workflow Engine(s)
Other Workflow Enactment Service(s)
Workflow Client Applications
Invoked Applications

Figure 2.1. WFMC's reference model for a WfMS

The process definition is a representation of a business process in a form which supports automated manipulation, such as modelling and enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process and information about the individual activities, such as participants, associated information technology applications and data, etc. [WfMC, 1999].

Modelling organizational procedures is a core concept in WfMS. Even further, if it is requisite to be flexible and support all kinds of human activities in the office, it is also necessary to be able to model human behaviour, as the WfMS' intent is to orchestrate their tasks. According to Nutt [1996], "a model is an abstract representation of a target phenomenon and represents a subset of the characteristics of the target system, i.e., it uses specific characteristics of the target system while ignoring others." Even though we recognize the intrinsic limits of the model, it is important to focus on the perspective that models should be as complete as possible to increase their applicability reducing the number of raised exceptions.

Recognizing the above mentioned limitation, Sheth et. al. [1996] introduced the concept of Work Activity Coordination as a multidisciplinary research to understand the interaction between technology, organization and human participants to cope with situations that are complex and dynamic, to learn how to adapt to frequently changing processes or to heterogeneous environments possibly involving multiple, dynamic and virtual organizations. They emphasize the multidisciplinary facet because these problems go beyond the current technological thinking and should involve other areas such as Organizational Sciences.

The acronym WfMS has been used with different meanings by the researchers in the field and some clarification is therefore required. In our conceptual approach, we emphasize the usage of the model as a script guiding actors through the required tasks to implement a business processes. In some systems, the flow of information and work among participants is not defined a priori and users define the next operator that should carry on with the work on the fly, according to their requirements. These systems, referred in the literature as ad hoc workflow [Georgakopoulos et al., 1995], do not have the semantics of the business process they model and do not play the role of scripts guiding actors on every action that should be implemented. Our approach is based on systems that support users according to this paradigm and therefore ad hoc workflows are out of the scope of this thesis.

Some core concepts related to the WfMS model consistency and correctness must be defined because they are used throughout this document [van der Aalst, 2001]:

- **Model consistency** – a model is consistent if it is capable of executing from its starting task to the final task without leaving any task in execution, i.e., there are no dead-locks or live-locks. Two additional conditions for consistency are to assure that every task has the chance to be executed if the proper conditions are met, and that the same task cannot be triggered twice at the same instant [van der Aalst, 2001];

- **Model correctness** – a model is said to be correct if it reflects adequately the process. It does not mean, of course, that the process itself reflects in all its essence the conditions users found when executing the process. It is only a stated relationship between the process and the workflow model that the system will follow.

Some generic model constructs defined in the WFMC's terminology and glossary [WfMC, 1999] are used throughout this document:

- AND-Split – a point within the workflow where a single thread of control splits into two or more threads which are executed in parallel within the workflow, allowing multiple activities to be executed simultaneously;

- AND-Join – a point in the workflow where two or more parallel executing activities converge into a single common thread of control;

- OR-Split – a point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches;

- OR-Join – a point within the workflow where two or more alternative activity(s) workflow branches converge to a single common activity as the next single step within the workflow. (As no parallel activity execution has occurred at the join point, no synchronisation is required.)

The concepts of pre-condition and post-condition are also relevant for this thesis [WfMC, 1999]:

- A *pre-condition* is a logical expression which may be evaluated by a workflow engine to decide whether a process instance or activity within a process instance may be started;

- A *post-condition* is a logical expression which may be evaluated by a workflow engine to decide whether a process instance or activity within a process instance is completed.

Finally, it is also important to distinguish between the different types of data manipulated by a WfMS. This dissertation uses the definition presented in the WFMC terminology and glossary [WfMC, 1999]:

- Workflow control data – data managed by the WfMS and/or workflow engines. Such data is internal to the WfMS and is not normally accessible to applications. This data is used to store information about running instances and the workflow state, e.g., the state of the instances, the users that are executing tasks and the starting time;

- Workflow relevant data – data used by the WfMS to determine the state transitions of a process instance and manipulated by the workflow engine and applications. For example, the cash value of a client is workflow relevant data if an instance can only proceed when the cash amount is above a specific value;
- Application data – data that is application specific and not accessible to the WfMS.

### 2.1.2. Model for a WfMS supporting unstructured activities

To account for unstructured activities, we must extend the reference model defined by the WFMC as shown in Figure 2.2. The main idea behind this extension is not to specify the interface details for WfMS interoperability as in the WFMC reference model, but to identify the new architecture required to support unstructured activities in a traditional WfMS. Therefore, the interfaces are identified here, but the functionality will be described throughout the dissertation. The major goal behind this extension is that the system must be able to switch from model guidance to map guidance. This functionality requires direct interaction with the enactment services of WfMS represented by interface A. Another required functionality is the capacity to implement model changes on the running instances. These changes require access to the process definition tools (interface B) and to the enactment services (interface A) in order to identify the instances on which the change is to be applied.

In some situations it may also be necessary to suspend the execution of a process model, to reallocate a task or to monitor system evolution using the standard WfMS functions. These features are implemented using interface C with the Administering and Monitoring Tools shown in the figure.

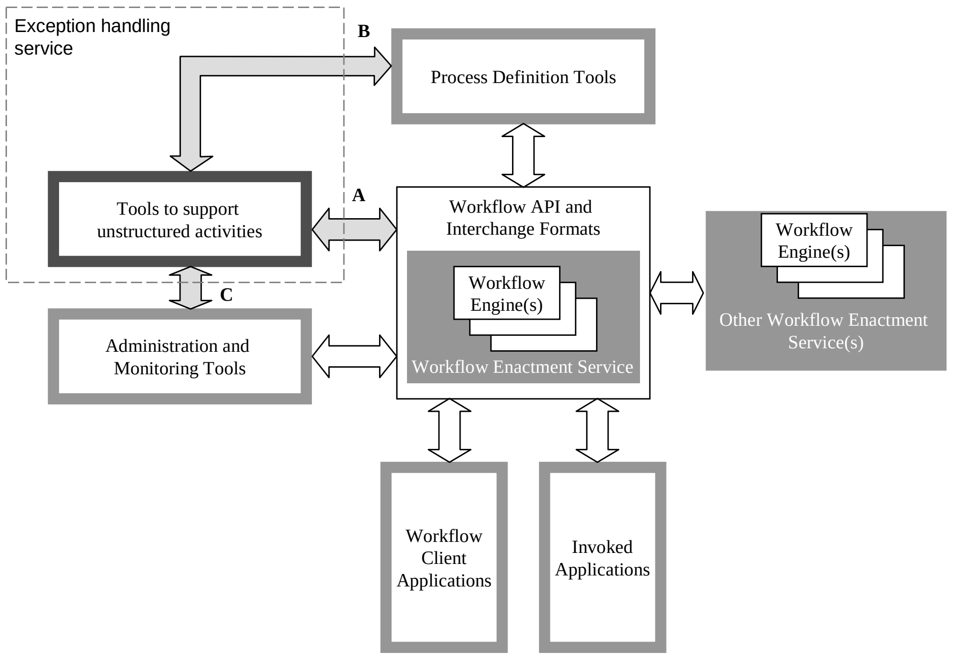

Exception handling service
B
Process Definition Tools
A
Tools to support unstructured activities
Workflow API and Interchange Formats
Workflow Engine(s)
Workflow Enactment Service
Workflow Engine(s)
Other Workflow Enactment Service(s)
C
Administration and Monitoring Tools
Workflow Client Applications
Invoked Applications

Figure 2.2. Extended WFMC's reference model

## 2.2 Organizational Sciences perspective

Hatch [2006] claims that "organizations arise from activities that individuals cannot perform by themselves or that cannot be performed as efficiently and effectively alone as they can be with the organized effort of a group." Therefore organizations involve a group of people that work together for a common goal. In Galbraith's [1977] words, an organization is "composed of people and groups of people aiming to achieve some shared purpose through the division of labour and integrated by information-based decision processes continuously through time."[2]

[2] The author's work experience in a car manufacturing company contributed to the relevance of the Organizational Sciences perspective in this thesis.

Contingency Theory is one of the research lines within Organizational Theory studying different forms of organizing work and positing that effective organizations should adapt to their environment [Donaldson, 1996]. According to this theory, organizations tend to fit their internal subsystems to the environments where they operate. As Morgan [1997] points out, “organizations consists of interrelated subsystems of a strategic, human, technological, structural and managerial nature which need to be internally consistent and adapted to the environment.” We will discuss how the solution proposed in this thesis can be used to improve the organizational ability to fit the environment.

One of the first works on Contingency Theory was published by Burns and Stalker [1961], where the authors developed the concepts of organic and mechanistic organizations. The former characterizes an open and flexible style of management where norms do not usually play a strong role and hierarchical rules do not restrict users from communicating with different departments. This type of organization is suited to changing technological and market conditions. The mechanistic organization is characterized by rigid norms, procedures and hierarchy and is suited to stable markets and technological conditions. These two organizational types are at the end of a continuum, and organizations do not entirely lie on each of them. Even in the same organization some departments might show organic behaviour (e.g., research and development department), while others show more mechanistic (e.g., production).

Since this theory was firstly introduced in the sixties, researchers have pursuit the best way for aligning internal subsystems to the environment and numerous developments within this theory have emerged. However, none of them has gained predominance and even the involved researchers recognize that their theories have boundary conditions [Hatch, 2006]. Nevertheless, the theory has been used as the basis for organizational analysis or, as Morgan [1997] puts it, the "contingency theory and an understanding of organizational needs can provide the basis for an organizational analysis. The analysis helps us describe detailed patterns of organizational relations, and it shows us possible solutions to the problems revealed."

The solution developed in this dissertation enables organizations to cope with the whole spectrum of organizational activities, from unstructured to structured activities while keeping the internal organizational subsystems unchanged. The objective is to increase the capacity an organization has to fit to the environment without adjusting its internal subsystems. I.e., for an organization that is adapted to the environment and with internal subsystems consistent and placed on some point of the organic-mechanistic continuum, the objective is to increase the organizational capability to deal with environmental instability. The organization should be able to switch to an organic style whenever the environment requires and then back to its standard mechanic style whenever the situation is overcome.

In order to understand the impact of this perspective on the organization as a whole, one should discuss its impact on each of its subsystems. We will start by the Technological subsystem. We will discuss how the environment affects this dimension and through it the other organizational subsystems, such as the structure. Then we will discuss how the exception handling solution developed in this dissertation can affect the analysis and improve the organizational capability to fit the environment. Technology, in the context of Organizational Sciences, is defined by Hatch [2006] as the "tools, equipment, machines, and procedures through witch the work is accomplished." According to Perrow [1986] the merit of this analysis is that "it provides some independent leverage in constructing organizational typologies because it focuses on something more or less analytically independent of structure and goals." Perrow uses two dimensions to classify how the technology affects the organizational capacity to react to failures:

coupling and interaction[3]. Interaction is the way subsystems within the technological system interact with one another and the effects these interactions have on the global system. Systems are classified as having complex or linear interactions. In complex interaction, one failure can produce multiple effects on more than one subsystem and produce unexpected and incomprehensible results because the visible aspects of the system malfunction do not allow operators to understand what has happened. They often require experienced and highly trained personal to understand the cause of the problem by looking at its manifestations. On the other hand, in linear interactions, single or multiple failures have visible and easily understandable impacts on the overall system. In these situations, it is not so important to use skilled personnel to understand the root cause of the problem. Examples of systems with complex interactions are nuclear power plants and airlines, while systems with linear interactions are assembly plants and post-offices. The second dimension is the degree of coupling. If coupling is loose, there is time to react to failures by replacing equipment, supplies or personnel or by choosing alternative ways of running the system. If the degree of coupling is tight, then the response must be fast (possible due to high impact of the failure on the system and/or on humans) and none of the above mentioned safeguards may be used. In tightly coupled systems there is less waste in material buffers and usually energy efficiency is higher. These systems also promote rapid and centralized decision making, strict schedules, rapid changes to production schedules and immediate response to deviations. Examples of tightly coupled systems are flying airplanes and nuclear power plants while loosely coupled systems are universities and research and development organizations.

[3] The focus of Perrow analysis is on system failures and not on organizational efficiency and is in line with this dissertation focus on how organizations cope with exceptional situations.

For systems showing both tight coupling and complex interactions, an inherent paradox undermines their structural organization: for one side they should be based on centralized decision making (due to tight coupling) while for the other side they should rely on the operators knowledge of the specific equipment in order to understand what is going on (due to complex interactions). However, the overall system view that should govern centralized and critical decision making within a timely frame (normally minutes or even seconds) is not compatible with information exchange with those that have critical information and knowledge about the malfunctioning subsystems. We believe that in this kind of situations, the system proposed in the present dissertation would play an important role since it allows the involvement and cooperation between the operators close the malfunctioning equipment or task and those with the responsibility for the centralized decision making. The sharing of updated information and knowledge improves the capability to make better decisions to recover from the situation [Morgan, 1997; Galbraith, 1977].

We will now place the focus of our analysis on the organizational structure and how it is affected by the environment. It should be noticed that these two analyses are not independent from one another and that some aspects may look similar. Nevertheless, both of them highlight different aspects of the phenomenon that deserve some discussion. Since organizations are supported by the division of labour, they must implement some coordination mechanisms to guarantee the coherence of the particular tasks in the whole. Galbraith [1977] and Mintzberg [1999] studied organizational structures and their coordination mechanisms. Mintzberg posits that organizations seem to need only five types of coordination mechanisms: mutual adjustment, direct supervision, work standardization, results standardization and standardization of operator's qualifications [Mintzberg, 1999]. If the task is well known in advance, the organization can elaborate procedures and rules in advance, adopting a standardization strategy. The hierarchy of command is responsible for the direct supervision of operators, guarantying the output is within standards, informing operations of commands originated at higher management levels, and resolving any non-planned situations. The work standardization is used to minimize the amount of information flowing up and down in the hierarchy, since it defines how operators should react on the situations they encounter. Information flow is a very important topic in management sciences because every organization has a limited capacity to process it [Mintzberg, 1999; Galbraith, 1977].

Discussing the administrative principles at the beginning of the twentieth century, Fayol [1919] realized that subordinates should handle routine matters, leaving managers free to handle exceptions to rules and standard procedures. This observation has resisted over time and Galbraith [1977] contrasts rules to exceptions, where rules are the standardization elements of organizations that handle the repetitive events, while new and unique events are treated as exceptions that should be handled by a manager responsible for all the affected areas. Exceptions involve a process of information gathering about the situation and decision about the appropriate actions to carry out. Therefore, exceptions increase the amount of information flows upwards the hierarchy of command, the decision making by middle and high management, and also increases the amount of control flow downwards the hierarchy to implement the decisions [Mintzberg, 1999]. If the number of exceptions is too high, compared with the capacity the organization has to deal with exceptions, medium and senior management may become overloaded and delays will happen on information flows. In these situations, the organization may face a potential behaviour control problem, because status information about problematic tasks does not arrive to management staff on time and decisions are delayed. This is recognized as a critical factor limiting the organizations whishing to achieve high levels of performance [Galbraith, 1977].

It is important to realize the attention exceptions deserve to Organizational Sciences researchers. This perspective on exceptions has played an important role in our approach because we have the aim to study their impact on organizations.

As it was said above, standardization is only possible if the organization has enough knowledge about task details that allows preplanning. It follows that if task uncertainty is high, the number of exceptions increases and no proper planning can be defined a priori. Task uncertainty is defined by Galbraith [1977] as the difference between the information necessary to perform the task and the amount of information already possessed by the organization. The greater the uncertainty, the greater the amount of decision making and information processing. In less routine and more diverse situations, organizations tend to use mechanisms with feedback from the operations level to orient their decisions, as opposed to programming and planning. It is also realized that cross-functional mutual adjustments become more frequent as operators and supervisors tend to adjust their unplanned actions with other departments. This will increase the time associated with each particular task and the organization may therefore lose performance.

The solution proposed in this dissertation allows a coherent exception handling that implements mechanisms to involve the affected actors. The implemented collaboration mechanisms explores new ways to facilitate mutual adjustments and diagnosis with the potential for improving global awareness of the situation and supporting situation diagnosis and decision making. Even further, decisions about activities that will be implemented are instantaneously communicated to all involved actors. The solution regarded as an exception handling facilitator, aims to augment organizations ability to deal with exceptions and improve its performance.

Relating with the previous discussion about situated versus purposeful actions, it seems that task uncertainty is related with the fit between plans and the concrete organizational environment, i.e., when task uncertainty is high, situated actions should prevail. Mintzberg [1999] shows that work predictability and diversification are intermediate variables in defining the organization structure and the adopted coordination mechanisms. The usage of a WfMS as the prevalent coordination mechanism should be analysed according to these studies since they will be more effective in some situations than in others.

## 2.3 Exceptions in WfMS

There are several ways to classify exceptions in a WfMS, according to the different perspectives that are applied to the problematic situation. In the related literature, some orthogonal criteria for exceptions classification can be found [Saastamoinen, 1995; Eder and Liebhart, 1995; Casati and Pozzi, 1999; Mourão and Antunes, 2003b]. In particular, one may consider a system perspective and assume that an exception triggers an exceptional event in the system. On the other hand, some exceptions cannot be identified by the system and must be triggered by humans or external applications [Casati and Pozzi, 1999; Heinl, 1998].

In Section 2.3.1 classifications on exceptions are elaborated. The first one, based on the classification presented by Eder and Libhart [1995], classifies failures and exceptions from a systems perspective. This widely adopted classification is described in the next section. The second classification, described on Section 2.3.2, was presented by Saastamoinen [1995] and adopts an organizational perspective about exceptions. The dimensions highlighted by this second classification provide some important characteristics that will guide our solution. Finally, Section 2.3.3 compares both classifications.

### 2.3.1. Systems perspective on failures and exceptions

Eder and Liebhart [1995] characterize failures and exceptions according to a single dimension, encompassing two types of failures and two types of exceptions:

- *Basic failures* – associated with failures on the systems underlying the WfMS (e.g., operating system, database management system and network failures);
- *Application failures* – failures on the applications invoked to execute tasks (e.g., unexpected data input);
- *Expected exceptions* – events that can be predicted during the modelling phase but do not correspond to the "normal" behaviour of the business process (e.g., a customer reporting a car accident in a car rental process);
- *Unexpected exceptions* – when the semantics of the process is not accurately modelled by the system (e.g., changes in business rules or a change in the order processing of an important client.)

The Eder and Liebhart's [1995] classification distinguishes two major types of deviations from the standard execution of a WfMS: failures and exceptions. The former result from system malfunctions either within the WfMS and the systems that support it or within the applications that implement the various tasks, where the later result from semantic discrepancies between the model and the application environment.

The authors recognize [Eder and Liebhart, 1996] that the currently available techniques to solve system and application failures do not overcome every situation and therefore suggest an escalating concept to transform into exceptions the failures that cannot be resolved in the level where they occur. Techniques to react in case of basic and application failures are important characteristics of a WfMS, since they increase the reliability and availability. These techniques are discussed on the next chapter to understand how the system reacts to these events and to discuss their applicability and limits. They were also studied to understand their usage in our solution. However, they are systemic approaches in the sense that they use mechanisms developed and inserted into the system to maintain control without involving the operators. They do not take in consideration the business semantics and are therefore not in the main focus of this dissertation. We will therefore concentrate on the expected and unexpected exceptions.

As defined above, expected exceptions can be predicted during the modelling stage but do not correspond to the “normal” process behaviour. These situations are usually excluded from the work model in order to reduce complexity. However, some authors posit that mechanisms should be implemented to handle these situations because they may occur frequently [Eder and Liebhart, 1995; Casati, 1998; Chiu et al., 2001; Sadiq, 2000c; Luo, 2001] and cause a considerable amount of work to handle. For example, consider the example of a client reporting an accident in car rental company, the company has to reschedule all future rentals for that specific car until the car is repaired. The “normal” behaviour should have been the returning of the car to the company, as planned, while the accident corresponds to a deviation or an “occasional” behaviour: an expected exception.

Chiu [2000] combines the above view with another orthogonal characteristic described as *exception source*. The exception source can either be internal, when the exception is triggered by the system, or external when a user reports the exception.

A further classification of expected exceptions was developed by Casati [1998] and identifies four classes, according to the events that generate them:

- *Workflow* – triggered when a task or a process is started or ended, it refers to the execution of the workflow itself. E.g., a deadlock situation or a loop being executed more times than expected. These events are therefore synchronous to process execution;

- *Data* – identified within the task that generates an error condition. The data events, even though identified within a particular instance can affect a collection of instances (e.g., a trip being booked twice for the same client.) These exceptions are synchronous to workflow execution since they only refer to errors in workflow relevant data (cf. Section 2.1.1) that can not be used for workflow evolution[4]. If the error refers to application data operations, they will result into an application failure that is not considered in the present class;

- *Temporal* – triggered on the occurrence of a given time stamp. These temporal events may be further divided into: timestamps, periodic and interval. Timestamps occur when a given completion date associated with a task is not respected (e.g., a car rental not delivered on the agreed time); periodic events occur on a determined periodical sequence (e.g., every morning at 9:00); and interval events are associated to time constraints between 2 tasks. E.g., the maximum time allowed after task 1 finishes before task n starts. These events are asynchronous to process execution

[4] Casati [Casati, 1998] includes workflow relevant data and application data in the data exception class. However, as it will be discussed in Section 4.4, all application errors are handled in a coherent manner in the system proposed in this thesis. Therefore, a small adjustment has been made in this class for coherence purposes.

because their firing does not depend on the execution of any workflow activity;

- *External* – activated by external sources, e.g., the above example of a customer reporting an accident. These events are asynchronous to the workflow execution.

This classification was developed for expected exceptions, because it assumes that the detection of an unexpected exception is always external to the system. However, any of the above classes can result from an unpredicted situation even though the symptoms are expected.  In fact, expected exceptions may also result in an unexpected situation (e.g., the firing of a predicted timer signals the delay of a particular task but the reason for the delay is not the expected and the designed handling procedure is not applicable). The distinction must be based on the cause and not on the symptoms, i.e., if the situation that cause the expected signalling is unknown the handling strategy can not be decided in advance and the exception is unexpected.

Therefore, if during the exception handling procedure it is identified that the concrete scenario is not predicted, the expected exception should be substituted by an unexpected exception. Even though the condition that triggered the event is foreseen (it is programmed in the system) the concrete situation that originated the condition might not be expected and the handling must follow a completely different approach. The work developed by Chiu et al. [2001] proposes a multi-layered procedural approach where a set of previously defined procedures capture the exception and try to handle it. If the exception can not be dealt by the procedure it is propagated to the upper procedure level. If none of the predicted handlers are able to handle the event, it is propagated as an unexpected exception.

Finally, the definitions found in the literature for unexpected exceptions state that they result from inconsistencies between process modelling in the workflow and the actual execution [Casati, 1998]; they are mentioned to be consequences of incomplete or design errors, improvements or changes in the business manoeuvre or quality and customer satisfaction issues unknown during the modelling stage [Heinl, 1998]. From our point of view, any situation that is not predicted in the model and requires out of the box activities (unstructured activities) is an unexpected exception. As mentioned in Chapter 1, when the model is no longer applicable, then the situated characteristics of actions prevail over the prescribed ones and we face an unexpected exception.

In situations where unexpected exceptions occurs frequently, one should consider redesigning the workflow model, if it is out of date, or adopting different technologies based on collaborative work or metamodel workflow systems [Casati et al., 1999] – user intervention is required. Metamodel workflows (cf. Section 3.2.1) enable users to adapt the workflow model to adjust business requirements. In this statement, Casati et al. realize that other strategies, different from expected exception handling, are required when the frequency of unexpected exceptions is high.

According to Perrow [1999], the increasing complexity of the systems that we are able to develop and the associated difficulty in assuring linear interactions, increases the probability of occurrence of failures with manifestations that do not facilitate the diagnosis. This also increases the complexity of the handling procedure associated with the failure. The situation is even more problematic if, as mentioned in Section 2.2, the system is also characterized by tight coupling. These systems usually require centralized decision making which is in some sense contradictory to a problem solving activity required by a complex problem solving situation. Arranging collaborative work environments where information flows between the operators with the most adequate knowledge about the situation at hand and the decision makers may improve the quality of the decision and, therefore, the reaction to the failure. The myriad of examples presented by Perrow [1999] demonstrate that complex interactions may happen in any situation: they are not a specific characteristic of complex systems.

### 2.3.2. Organizational perspective on exceptions

Saastamoinen [1995] proposed a taxonomy based on the organizational semantics associated to exceptions. The taxonomy defines a set of base concepts necessary to construct a consistent conceptual framework that fundaments the characterization of organizational exceptions. The classes of organizational exceptions were then obtained from empirical studies. In order to understand and contextualise the classification, this section starts by describing the theoretical assumptions. It is followed by the taxonomy description.

According to Saastamoinen, it could be said that any situation for which the organization has no rule is an exception. This is the line of thought also adopted by this research. The author defines event as a piece of work to be handled by an office that is caused by a detected phenomenon or a state of the system. An event type is a specification of the common features found in certain events. The concept of rule also plays an important role in this organizational perspective and is the basis for exception definition. Rules are defined as a formal way of specifying a recommendation, a directive or a strategy, expressed as “IF premise THEN action” or “IF condition THEN action”. An event handling rule is defined as an orderly set of rules that precisely and accurately guide an actor handling certain types of events. An office procedure has a broader scope and is defined as an orderly set of event handling rules aimed at reaching a specified goal in the office by directing an entire event handling.

Once the association between rules and event handling rules is established, the concepts of normal event, main line event and variation event constitute the last step towards the exception definition. A normal event is an event with the necessary identifying and handling rules. A main line is an office procedure for the most predictable events of a certain type, and a variation is work that is added to the main line. Note that a variation is an office procedure for less predictable but still known events of a certain type. A main line event is a normal event that can be handled by the main line while a variation event is an event not handled by the main line but handled by another office procedure. The conceptual framework for exception definition as defined by Saastamoinen is now created. An exceptional event (exception) is an event that neither the main line nor the variation procedures can handle.

As discussed by Saastamoinen, the definition of rule is narrower than the variety of rules that exists in an organization, e.g., good business practices, precepts, regulations, conventions, principles, guiding standards, rules of thumb and even maxims. However, these kinds of rules are not precise enough to establish a consistent ground to serve as the basis for a framework, even though they represent the knowledge of the organization.

Saastamoinen developed the taxonomy using the above concepts as well as empirical studies carried out in an organization, with a special attention to the social and financial impacts of exceptions. Six different criteria were proposed to classify exceptions:

- Exceptionality – difference between the exceptional and “normal” event;

- Handling delay – time elapsed between the exception identification and handling;
- Amount of work – extra work required to handle the exception when compared to the normal event;
- Organizational influence – number of people involved in the exception;
- Cause – a measure of the importance of the reason for the exception;
- Rule impact – the changes in the organization's rules due to the exception.

Three classes of exceptions were identified according to exceptionality: established exceptions, otherwise exceptions and true exceptions. Established exceptions occur when the handling procedure for the event is defined but the rules in the organization do not support users identifying the correct one. Otherwise exceptions occur when the organization has rules to handle the normal event but do not apply completely to the case. Finally, true exceptions occur when the organization has no rules.

According to the organizational influence criteria, exceptions can also be classified at employee, group and organizational level. Employee exceptions are situations that affect only the work of one person. Group exceptions affect a group of people working within the same process, in the same kind of job or in the same process. Organization exceptions affect the work of persons in more than one department or project in the organization.

The criteria handling delay, amount of work, cause and rule impact are important to understand the organizational impact of the exception after the organization has regained normal behaviour. They are only established after the handling procedure is finished. Since they are of no use during the handling phase they will not be considered in the reminder of this dissertation.

The dimensions adopted in this dissertation to classify exceptions from an organisational perspective are:

- Exceptionality – difference between the exceptional and "normal" event;
- Organizational influence – number of people involved in the exception;

## 2.3.3. Comparing and integrating perspectives

By using different perspectives to classify exceptions, we may avoid a biased view or sensibility about the problem. Exceptions in WfMSs are events that must be handled by the system and/or the affected users. The users, whenever involved, should have a clear picture of the event in order to decide on the most suitable handling strategy. This includes understanding the system behaviour which may have lead to an exception. The system and organizational perspectives described in the two previous sections are combined in this dissertation to classify exceptions and guide both the system and the user reactions. Therefore, it is relevant to compare the descriptive capabilities of both classifications to understand their strengths and weaknesses. Table 2.1 lists different dimensions and the corresponding descriptive capabilities of the taxonomies.

From the table it can be seen that familiarity with the event is the only characteristic classified by both taxonomies. The remaining dimensions are particular to each taxonomy. Organizational impact of the exception is only classified by the organizational taxonomy while detection and type of event are only classified by the system taxonomy. Since both taxonomies seem to be complementary in all dimensions except one, the table is also useful to identify the dimensions that should be used during unstructured activities support: *familiarity with the event*, *organizational impact*, *detection* and *type of event that generate the exception*. This classification is used to support users.

Table 2.1. Comparing system and organizational perspectives on exceptions

| | System | Organizational |
|---|---|---|
| Familiarity with the event | Some | Rich |
| Organizational impact | None | Rich |
| Detection: manual or automatic | Rich | None |
| Type of event that generate the exception | Rich | None |

System taxonomy uses two classes for the familiarity with the event dimension, the expected and unexpected, where events fall into one of these extremes. They are either expected and the organization should have rules to handle them, or they are unexpected and no rules exist. By using three classes, the organizational taxonomy has more descriptive capability to classify the organizational familiarity with the event. Class established exception means that the handling procedure is defined for the event but it cannot be found. When the organization finds the appropriate handling procedure the handling procedure is defined. Otherwise exceptions may involve adjustments to organizational rules to fit the situation. For true exceptions the organization has no rules. These dimensions in both taxonomies also measure the capability that the organization has to deal with the event. However, they do not have enough descriptive capability that enables us to identify if unstructured activities are required to handle the event as a problem solving activity. It may be the case that just readjusting some tasks is enough to overcome the situation, e.g., if new legislation requires that for loans above a certain amount the clients should have goods in the value of at least 40%, the organization would have to reprogram its tasks in order to assure this new condition.

The following section describes a new taxonomy that aims to support users identifying if the handling of the event should involve unstructured activities.

## 2.4 New exception classification

In this section we discuss an extended exception classification focused on the knowledge about the situation processed by the organization and on the planning capacity. We start by the former dimension and finish with the later. In Section 2.3.1 exceptions were classified as expected or unexpected. However, a concrete situation may not entirely lye on each of these classes. Instead it may be similar to an expected situation but not completely equal and the procedure may have to be adjusted. On the other hand, a situation that is new to the organization (unexpected exception) may have some aspects similar to a previously event. In summary, in this thesis we advocate a novel approach to exception classification, assuming a continuum from expected to unexpected exceptions and integrating the system and organizational perspectives.

Figure 2.3. shows the expected-unexpected continuum with the Eder and Liebhart's [1995] taxonomy bellow the line and our proposed taxonomy above the line. In our taxonomy, we propose three exception types. The definition of these types is based on the similarity degree of the situation with the complete set of rules and past experience that exists in the organization. *True expected* exceptions are at the expected limits of the spectrum and are those for which the handling procedures are entirely defined. For *Extended expected* exceptions, which initiate close to the expected limits and extend into the spectrum, some guiding behaviour can be drawn from rules and past experience even though some adjustments are required. A matching technique may be defined and used with the organizational knowledge base of previous exceptions to find the most similar event and analyse the similarities and the differences to the actual situation [Luo et al., 2002; Hwang et al., 1999; Klein and Dellarocas, 2000; Grigori et al., 2001; Weber and Wild, 2004; Adams, 2007]. Then the reaction activities can de drawn based on the activities that were applicable to the matched event by adjusting them to the new situation. Finally, *Effective unexpected* exceptions are those for which the organization can not derive any guiding behaviour from the organizational knowledge base. Since the system may not obtain any handling procedure, the user involvement is mandatory. Even further, some exceptional situations can represent a strategic opportunity that will not be recognized by the system (e.g., a user complaining on some non-implemented feature in one good and when the sales representative is talking to the user he identifies that with minor changes to the equipment it can solve some other pertinent problem).

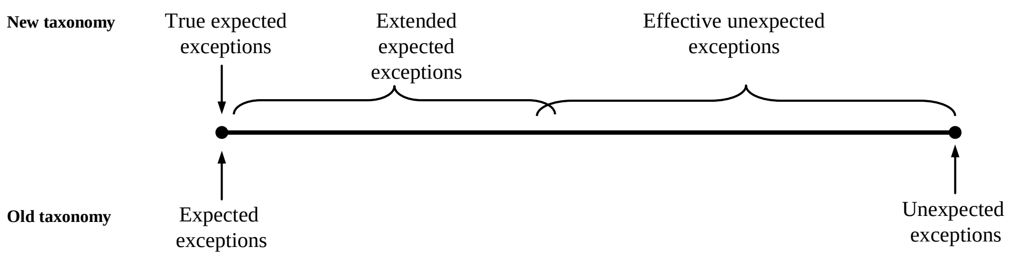


Figure 2.3. Three exception types in the expected-unexpected continuum

To overcome effective unexpected exceptions, WfMSs should not rely only on the existing documented organizational knowledge in the form of procedures represented as models inside the system. Operators should be provided with the necessary mechanisms to react in a collaborative way and decide the best solution for the particular case. Also, according to the discussion in the Chapter 1, unstructured activities should be supported by this type of systems.

A new dimension will now be used to further classify effective unexpected exceptions: the planning capacity for the handling procedure. In this dimension two classes are identified:

- *planned effective unexpected exceptions*;
- *ad hoc effective unexpected exceptions*.

For *planned effective unexpected exceptions*, a reaction plan can be established before the reaction starts. It usually means the organization has enough knowledge about the situation to establish a reaction plan (e.g., a new legislation that the company has to comply within a determined period of time). For *ad hoc effective unexpected exceptions* no plan can a priori be established. The reason may be that there is not enough knowledge about the event that enables advanced planning of reaction procedures, or the environmental conditions vary so much that no plan can robustly be defined. In these situations the situated characteristics of actions should prevail over prescribed ones (cf. Chapter 1) and the reaction must be implemented in an ad hoc way (unstructured activities) involving problem solving among participants both for situation diagnosis and recovery. For example, if a truck with a very important delivery is stuck on traffic jam users can not define a priori what is the best action to overcome the situation. It may be the case that traffic just starts to flow and no reaction is necessary, while in some situations another delivery by a different road may be the best solution. Users should collect as much information as they can and react as the situation evolves.

Therefore, ad hoc effective unexpected exceptions require human intervention and an innovative posture from the organization to deal with the situation. As no plan is available, human reaction should be map guided, according to Suchman's definition [1987]. This exception type is the main focus of the present dissertation.

From now on, they will be referred as *ad hoc effective unexpected exceptions* or simply *unexpected exceptions* when no distinction is necessary.

To effectively implement map guidance, human operators should be fed with valuable information about the environment and workflow status so they can decide their actions. On the other hand, the situated characteristics of the actions also require that users should not be restricted by any model that was previously defined. We have just introduced the openness and the completeness requirements discussed in the next section.

After the 9/11 event, officials started to write procedures for clearing the skies. After some effort, they realized that they would better rely on the judgement of their controllers and managers. It seems that even after the event, it was not easy to write the handling procedure, meaning the event was an ad hoc exception.

On the other hand, at the very start the event seemed to be an expected exception. However, operators changed their perception during the event and exception classification was changed to unexpected.

## 2.5 Openness and completeness

This section describes the two requirements we consider mandatory to implement the support to unstructured activities: openness and completeness.

> *The openness requirement states that the system should be able to collect environmental and workflow status information to support users on their map guided activities. Users should then be able to look*

> *for the most relevant information to understand the situation and to decide on the most adequate activities to carry out.*

On the other hand, users should not be restricted to the services provided by the exception handling system. The challenge is to manage awareness and consistency with the exception handling activities carried outside the WfMS scope. Our solution integrates environmental information about external activities but will not assume control of those activities. The main idea is that unstructured activities characterize human reactions to effective unexpected exceptions.

Since some operations are carried outside system boundaries, it should be possible to maintain information on such activities and register any relevant information that should be useful to involved actors. This requirement is also important to maintain an update history log of the implemented activities carried out during the exception handling procedure.

> *The completeness requirement states that an exception handling system should consent users to carry out recovery actions without restrictions, i.e., the flexibility of the exception handling system should be on par with the flexibility actors have on their daily activities when working without system control.*

This definition is based on the notion that people tend to solve their problems with all the available means. If any system restrictions are imposed to the users' primary goals, they will overcome the system [Strong and Miller, 1995; Hayes, 2000].

It is important to note that flexibility implied by the completeness requirement should be supported by the WfMS enactment service (cf. Section 2.1). When the system is running, the enactment service is responsible for instantiating the defined models and guarantee that the processes run according to what is defined.

Therefore, any deviations from the standard procedures must be implemented at this component.

The consequences of this open perspective on WfMS are profound. For instance, the restrictions to the common model changes, described in Section 3.2.1, must be relaxed [Rinderle et al., 2004a; van der Aalst and Basten, 2002; Agostini and De Michelis, 2000b; Ellis et al., 1995; Casati et al., 1996]. These restrictions are only applicable if one wants to keep the execution under the specified work models. However, if the objective is, for instance, to graciously abort a workflow instance, no consistency check is necessary. Even further, if the user decides to implement a recovery action that deliberately inserts structural conflicts in the work model, s/he should be advised on potential problems but allowed to carry out that action.

Consider for instance our 9/11 motivating example. Facing the exceptional event, air traffic controllers tried to do whatever they could to overcome the situation. They used any available means to fulfil their goals and established their goals on the fly as they were collecting information about the situation. No system could have been designed to model the user's reaction on situations like these, simply because they were never considered possible.

The restrictions that air traffic control operators have on their daily operations to keep system consistency had to be overcome. Take into consideration a common situation where a plane has to be rerouted due to a storm: (1) the air traffic control operator contacts the pilot to arrange an alternative route suitable to reach the destination within the plane's fuel capacity; (2) a new route is then agreed; and finally (3) the new process model for the affected instance is adopted and system consistency is maintained. However, these restrictions were not taken into consideration when the order to land all planes was issued on 9/11 at 9:45AM: they would land on the closest available airport or being kept on hold until ground space was found, while giving priority to planes with fuel problems.

We emphasize that although model guidance could not be adopted on 9/11, map guidance was apparently considered beneficial: the FAA command centre in Herndon, after the second plane hit the south tower, decided to start writing on a white board information regarding all planes across the country suspected to be under hijacker's control. This situation also stresses the role of monitoring information and external tools in map guidance.

## 2.6 Summary

In this chapter we have proposed a new reference model for WfMS that is capable of handling any type of exceptions. A new component was identified together with the interfaces with the WFMC reference model components.

Then, our framework was discussed in the light of the Organizational Sciences perspective in order to help us understanding the impact of the system in the organization.

Existing exception taxonomies were revised and the dimensions that are used by the solution to support unstructured activities were identified: familiarity with the event, organizational impact, detection, and type of event that generate the exception. Then, a new classification was derived to help identifying if the event requires unstructured activities support.

Finally, the mandatory requirements the system must implement to support unstructured activities were established: openness and completeness.

.

# Chapter 3

# Resilience in WfMS

Since WfMS support business processes, it is very important that they keep operational during business operations even under unpredictable situations. Their ability to adjust to actual businesses solicitations and to react to different hazardous conditions such as failures and exceptions is a core property for a WfMS to actually support organizations.

> *The resilient property of a WfMS concerns its ability to maintain a coherent state and continue supporting business processes after being subject to any hazardous situations that affect its execution.*

It should be emphasized that this is a runtime property of the WfMS, because predicting any possible causes of failure or exception during design is considered very difficult or even impossible and makes the system very complex and hard to manage [Eder and Liebhart, 1998; Dayal et al., 1990; Casati, 1998; Klein and Dellarocas, 2000; Mohan et al., 1995]. The strategy to manage failures and exceptions is to increase system resilience. Resilience requires both robustness, to avoid system crashes due to failures, and flexibility to adjust to deviations on the user and organizational conditions.

This thesis focuses on WfMS support to ad hoc effective unexpected exceptions (cf. Section 2.4). *Flexibility* is a core concept to deal with these events and should be on par with the flexibility actors have on their daily activities when working without WfMS support (cf. completeness requirement Section 2.5). A good compromise between flexibility and robustness is an important characteristic of a resilient WfMS [Nomura et al., 1998]. Robustness is important to keep the organization under control. The main objective should then be to increase flexibility without loosing all the advantages of the WfMS model and coordination support. This chapter is dedicated to analyse the most relevant developments on this field to increase robustness and flexibility: resilience.

Since the early developments of WfMS (the research field was then known as Office Information Systems), in the seventies, researchers have focused on the objective to specify high level programming languages that would adjust easily to business changes [Hammer et al., 1977; Ellis and Nutt, 1980]. On the other hand, supporting users in organizations also requires systems to work on heterogeneous, autonomous and distributed environments (cf. Section 3.1) [Worah and Sheth, 1997; Bussler, 1999]. Therefore, researchers had also to develop techniques to improve WfMS robustness in these environments. However, during the nineties, it became clear that the systems based only on high level programming languages did not achieve the *de facto flexibility* demanded by users. WfMSs did not experience the market acceptance the researchers were expecting and the main appointed reason was their lack of flexibility [van der Aalst and Berens, 2001; van der Aalst et al., 1999]. Research work was invested on this issue. Many different approaches have been experimented as different research groups had different understanding on the type of flexibility that would trigger market acceptance. The techniques to **increase robustness and augment flexibility** have been grouped in this chapter because, even though they highlight different facets of WfMS, they are also strongly related. Increasing flexibility can have a negative effect on robustness and vice versa, and we need both to accomplish resilience.

The next section is dedicated to analyse the systemic approaches to resilience. The systemic techniques assume the objective to provide the WfMS with the necessary mechanisms to react to basic and application failures, and to expected exceptions. Systems to handle expected exceptions are inserted in this group because they do not increase flexibility when users face a new exception at runtime. Special modelling constructs are used to augment the model's applicability and their main advantage is to increase systems robustness when coping with predictable situations.

The second section proceeds with the human oriented approaches to increase resilience. The various research lines were grouped in four classes according to their approach to the problem (classification inspired by [Han et al., 1998]): metamodels, open-point, other approaches and supporting unstructured activities.

The following section compares the different approaches to augment resilience and draws conclusions to use in the remainder of the thesis Finally, the last section is dedicated to discuss the expressiveness capability of modelling assumptions and their impact on implementing flexibility. In this section the metamodel assumptions adopted in this thesis are presented together with the operations available to change the workflow model during enactment.

## 3.1 Systemic approaches to increase resilience

We have previously distinguished (cf. Section 2.3.1) system and application failures, where system failures result from malfunctions either within the WfMS and the systems that support it and application failures result from errors in the applications that implement the workflow tasks. Systemic approaches aim to handle this type of events and are defined as:

> *Systemic approaches are designed to handle failures and exceptions without human intervention.*

However, it has been recognised by the researchers in the field that in some situations it is not possible to handle the event without human intervention [Eder and Liebhart, 1996; Casati, 1998; Chiu, 2000]. A propagation mechanism must be foreseen to transform these situations into unexpected exceptions so they can be handled with human support. Systemic approaches are therefore limited by the limited capacity of WfMS to overcome problems without human intervention.

It should be noted that WfMS usually operate in a heterogeneous, distributed and autonomous environment [Worah and Sheth, 1997; Bussler, 1999] and the designed solutions should take this aspect into consideration. Heterogeneous, because tasks run on different settings, ranging from pure transactional (e.g., database systems) to completely non-transactional environments (e.g., making a phone call) and have to integrate legacy applications built according to different computing paradigms. Distributed, because tasks run on different locations (e.g., on computers close to user's location) throughout the organization, and with autonomy, since each task runs on its environment using the available data and resources.

### 3.1.1. Failure handling

WfMS use a database management system (DBMS) to manage workflow relevant data. Transaction processing techniques, developed in the DMBS field, guarantee data integrity and consistency on system failures. In fact, most of the commercially available DBMS on the market implement the necessary transaction processing mechanisms to react in case of failure, returning the system to a coherent state and enabling forward execution [Casati, 1998]. Therefore, on the event of a system failure, the DBMS implements a standard failure handling task by restoring a previous coherent state. The WfMS is then able to proceed with forward execution.

Alonso et al. [Alonso et al., 1994; Alonso et al., 2000], realize that when the WfMS spans over a wide area network involving several thousand users, hundreds of thousands of concurrently running processes and several thousand of sites, fault tolerance is necessary to increase robustness and availability in case of serious failures. This discussion will not be further developed because this type of approach is out of the scope of this thesis. As mentioned before, we assume that, if the system can not recover from the failure, it is propagated as an unexpected exception.

We will thus focus on application failures because they usually have impact on the task and are more difficult to handle, considering their associated business semantics. Application failures must be handled by a completely different approach than traditional DBMS, because a typical task in a WfMS spans over long periods of time – long-running activities [Dayal et al., 1990]. If isolation and atomicity properties of traditional transaction processing systems are enforced, the level of concurrency and task cooperation required by WfMS are compromised. Even further, applications may not run on transactional environments since a task can be a person making a phone call, a meeting or filling a spreadsheet [Alonso et al., 1996c]. Advanced Transaction Models (ATM) with relaxed Atomicity, Consistency, Isolation and Durability (ACID) properties were proposed to overcome these problems [Georgakopoulos et al., 1995; Jin et al., 1993; Chen and Dayal, 1996]. For instance, by relaxing the isolation property, other tasks are able to access data before a transaction finishes. Compensation tasks are defined for each committed task to allow backward recovery and restoring data consistency and correctness, and to proceed with forward execution. However, experiments with ATM showed scarce applicability because they have limited ability to model the rich organizational contexts where WfMS usually run, which are characterized as heterogeneous, autonomous and distributed [Worah and Sheth, 1997; Breitbart et al., 1993; Alonso et al., 1996a]. ATMs rely on traditional transactions, with enforced ACID properties, as the building blocks of larger transactions with some relaxed properties. I.e., the building blocks are pure flat transactions, and after each building block is committed, some properties are relaxed. According to Alonso et al. [1996a], these solutions are biased from a DBMS view of organizational procedures which may result in a restrictive model capability. Worah and Sheth [1997] emphasize the need to look beyond transactional features, as they are only a small part of the workflow application domain.

Although recognizing ATMs limits on WfMSs, it is important to emphasize they have a strong theoretical basis to assure data consistency, correctness and recovery on the event of failures when tasks run in transactional environments. This research trend was very important during the 90s when some important solutions were proposed [Worah and Sheth, 1997]. In the following, we will describe some ATM based approaches.

The ConTract model [Wachter, 1991] defines the notions of scripts and steps. A ConTract is composed by a sequence of steps executed in a transactional environment and by scripts that specify the control flow relations between steps.

Each step is programmed independently from the workflow sequence and is responsible for the execution of one activity. Isolation and atomicity are relaxed between steps, meaning that other tasks/transactions have access to the task resources after completion. Scripts also specify *compensation activities* that may be carried out for recovering each committed step. E.g., to enable backward recovery, it is necessary that all steps already executed and committed to the database be semantically compensated by an activity defined in the script. Workflow continuation is then assured after the system is back in a consistent state.

Dayal, et. al. [Dayal et al., 1990; Dayal et al., 1991; Chen and Dayal, 1996] introduced the notion of *spheres of control*. An activity can either be an atomic task or a tree with other atomic tasks. A failing task in a sub-tree may be compensated using the mechanism previously described. However, if the compensation fails, the error is propagated upwards the hierarchy where a new compensation is tried on the parent task. All subtasks specified by the parent are aborted before compensation is started. The highest node where the compensation is successful represents the scope of the roll-back operation and is the starting point of the forward execution. Dayal et al. also introduced *Event Condition Action* (ECA) rules to detach the detection of a failure from its handling. This approach increases the flexibility associated to the failure handling process, since rules are data driven and can be triggered when some important event is detected. The handling routine can run in the same or in a different context of the application that originated the event. The system is also easier to change since it is not always necessary to change the applications: changing the rule may be enough.

Compensation spheres were introduced by Leymann [1997]. They represent a collection of activities such that either all activities run successfully or all activities must be compensated. As in the previous approach, compensation tasks are defined for each one of the elementary tasks. It is also possible to reinitiate the affected branches at the entry points of the compensation spheres, where the user can specify whether compensation activities should be executed, whether some administrative work should be carried out, or whether the flow should proceed at the entry point without any compensation. The model is also based on the notion of atomic spheres, where ACID properties are reinforced for all tasks within the sphere.

The notion of atomic spheres was also explored by Hagen and Alonso [2000]. As in Leymann’s work, atomic spheres have to be successfully executed or they are undone in the occurrence of a failure. Each atomic sphere may relax one of the traditional transaction properties of atomicity, isolation or consistency. In spheres of atomicity, three different levels of isolation are considered: atomic, quasi-atomic and non atomic. Atomic tasks run on traditional ACID transaction environments and have no effect if they fail. For quasi-atomic tasks, a compensation activity has to be defined to undo their effects, whereas non atomic tasks can not be eliminated. Error handling is defined at the sphere level. This model uses the programming notion of exception handling introduced by Goodenough [1975], which separates exception detection from exception handling. Handlers are embedded in the atomic sphere and execute the recovery activities when the failure is detected.

The approach proposed in WAMO [Eder and Liebhart, 1995; Eder and Liebhart, 1998] is similar to the previous one, but the model is enriched with the notion that tasks fall into one of three categories:  1) tasks that are compensated with or without side effects; 2) tasks that do not need to be compensated; and 3) tasks that can not be compensated. A new type of task, marked as “force”, enables the WfMS to keep trying task execution until successful. This type of tasks is usually not problematic, e.g., printing a document. When the compensation mechanism is initiated, tasks are undone until a decision point is reached and forward execution may proceed. User intervention is required for tasks marked as “force” that are not successfully executed after several retries, and for failures on tasks that can not be compensated. This human intervention, performed at the task level, can be used to unblock any situation regarding task execution (e.g., a print task is not accomplished because there is no paper in the printer) or compensation activities that require humans (e.g., make a phone call to cancel an order already issued to a supplier).

Finally, a similar but more flexible approach is proposed by Kamath and Ramamritham [1998]. In their work, a failing task may initiate a complete or a partial compensation. The former forces all the context to be compensated, as in previous approaches, while on the later only the failed task is compensated, and forward recovery is carried out by an incremental execution of the task (e.g., with new inputs obtained from task re-execution). With partial compensation, not all the tasks within the compensation sphere have to be compensated like in the previous solutions.

The handling of application failures based on transactional approaches offer, in general, extreme and expensive solutions in terms of lost work [Alonso et al., 1996b; Worah and Sheth, 1997], because all tasks in the sphere have to be compensated. If business semantics is taken into consideration during failure handling it may be possible that not all tasks require compensation. Therefore, some application failures should be handled as expected exceptions [Casati, 1998].

### 3.1.2. Handling expected exceptions

Dayal et al [1990; 1991] recognized the rigid compensation policy of the ATM approaches and proposed the ECA rules to increase flexibility. Casati et al. [1999] proposed a system based on detached ECA rules. Detached rules run in a different context, i.e., the action part of the rule, responsible for the exception handling procedure, runs in a transactional context different from the task where the exception was originated. According to the author, in most cases rules do not need immediate handling and should not interfere with normal process execution. The language Chrimera-Exc [Casati et al., 1999; Casati, 1998] was developed to specify ECA rules and augment the WfMS modelling capability to detect and handle expected exceptions. Each rule can monitor multiple events categorized in four classes: data manipulation, external events, workflow events and temporal events (these classes were described in Section 2.3.1.) The action part may execute several primitives belonging to two categories: data modification and workflow management. The former stands for operations related to object management, such as create, modify or delete; while the later are workflow related functions such as notifications to agents, starting new tasks, cases, sub-processes or reassigning tasks to different agents.

ECA rules are also used in ADOME [Chiu et al., 2001; Chiu, 2000] to specify exception handlers in an object oriented WfMS system. A sequence of rule sets is defined and the exception is evaluated on each set until resolved. The first set is composed by mandatory rules, which must always be executed (if defined) whenever the corresponding event is triggered and the condition evaluates to true (e.g., inform manager about the event). If this set does not solve the situation, the exception is propagated to the second set. The second set is specific for each task and particular application context. Since they are specific for a particular task that should produce outcomes, they are evaluated first. The third set starts from the current activity and follows up the task hierarchy to identify any applicable condition to the situation. The WfMS has also built-in generic exception handlers that are applicable to all activities (e.g., if the resource for a particular task is not available, the WfMS tries to replace it). If none of these four sets of ECA rules resolves the exception, the system tries to re-execute the tasks marked as repeatable, skips tasks marked as optional or tries another path for tasks marked as replaceable. Tasks marked as critical, without specific exception handlers, are classified as unexpected exceptions. Finally, if this last step does not solve the situation, the exception is propagated to an unexpected exception and human involvement is requested. However, human involvement is limited to a list of solutions proposed by the system and unstructured activities are not supported.

Luo et al. [2002] describes a system using a Case Base Reasoning to extend exception handling. A case repository is maintained with information about previous exceptions and handling procedures. When an exception is detected, the *intelligent problem solver* is consulted and similar cases are retrieved using similarity measurements weighing the exception description, the workflow model and the context. By using similarity reasoning, the system enlarges the traditional notion of expected exceptions. Nevertheless, the authors mention that human intervention is required whenever there is no matching.

The system purposed by Vojevodina et al. [2005] uses an inferring mechanism to derive the most adequate strategy to handle the situation. Special effort is placed on the event classification and the three basic exception handling functions of detection, diagnosis and monitoring discussed in detail. The diagnosis must be finished before the handling procedure is initiated. Since user intervention is not foreseen, the system relies on the organizational knowledge base to conduct the exception handling procedure.

The system proposed by Weber and Wild [2004] uses Conversational Case Base Reasoning to implement just-in-time updates to the workflow model. When an exception is detected the system uses the knowledge base of previous events and collects the cases with similar characteristics. The user then verifies the case conditions to compare with the specific event that has to be handled. After choosing the most adequate case, the handling procedure can be derived. When this new exception is handled a new case is inserted into the system. When the amount of cases relating a particular context becomes significant, the cases are abstracted into new model rules. The system has therefore a learning capacity. However, the user is constrained by the existing system knowledge and no unstructured activities support is foreseen. The system is designed to handle extended expected exceptions (cf. Section 2.4).

Klein and Dellarocas [2000] also proposed the construction of a knowledge base of exceptions related to the processes and procedures that may handle them. A taxonomy of processes is constructed in a way that the generic processes are at the top and leafs are specific processes. When an exception is raised, a matching mechanism identifies the handling procedure that best fits the situation. Again, this technique has the objective of augmenting the system capability to handle expected exceptions.

Some other approaches use exception mining techniques to analyse, predict and prevent the occurrence of exceptions. In [Grigori et al., 2001], the authors describe a system which analyses businesses models and execution logs to extract knowledge about the occurrence of exceptions. This knowledge is then used to improve the model or to make organizational changes (e.g., if the system realizes that a process is late when a given supplier is involved the organization may change the supplier.) In [Hwang et al., 1999] the authors propose a system that scans over the previously detected unexpected exceptions and suggests the adopted solutions.

Considering the exception continuum defined in Section 2.4, all these systems are designed to handle true expected exceptions and extended expected exceptions. They are designed to increase the system robustness for these exception types since they try to maintain control in the system. The system proposed by Chiu [Chiu, 2000; Chiu et al., 2001] supports user involvement in unexpected exceptions by suggesting a list of possible solutions for the case. If the solutions are known to the system the exception is therefore an extended expected exception. Weber and Wild

[2004] also consider user involvement in the handling procedure. Users are supported using the knowledge obtained from previous experiences and therefore relevant also for extended expected exceptions. The flexibility of these approaches is limited since they only account for the knowledge inserted into the system. Even when the system has learning capabilities, they are grounded on the existing knowledge, and does not support complete new approaches – unstructured activities, as defined by our approach, are not considered.

The solutions described in the two preceding sections should be implemented by the WfMS to assure system robustness, but they are not very relevant for the system developed in this thesis because they do not support unexpected exception handling. Nevertheless, they were investigated to identify opportunities of their usage on the proposed system. Of course, all this are low level system approaches and their applicability is not as immediate as some of the higher level approaches found to handle exceptions.

## 3.2 Human-oriented approaches to increase resilience

As mentioned in this chapter introduction, resilience is a system's runtime property. Therefore human-oriented approaches to increase flexibility may be defined as:

> *Human-oriented approaches are designed to support human interventions in business processes at runtime, and increase the systems' resilience by increasing its flexibility.*

Flexibility is related to the operations not predicted in the model that users carry out during the execution phase of the system in order to accomplish work [Agostini and De Michelis, 2000a; van der Aalst et al., 1999; Ellis and Nutt, 1993; Casati et al., 1996]. I.e., when a process is instantiated, the user may implement some operations not predicted in the model but made available by the workflow enactment service. When the intervention requires model adaptation, the process definition tools (cf. Section 2.1) may be used to design the new model, while the enactment service replaces the old model by the new and continues operation.

The type of operations available to users depends on the features implemented by the enactment service. Various approaches to flexibility can be found in the literature, as different authors have particular understandings on the most important inhibitors operators face on their daily operations.

We start by identifying the major inhibitors to flexibility and proceed with a brief summary on how the solutions to overcome them were implemented.

The reasons for the lack of flexibility in current WfMS are:

1. Inability applying model changes to already running instances [Rinderle, 2004b; van der Aalst and Basten, 2002; Ellis et al., 1995]. If some business requirement demands model adaptation (e.g., due to a legislation change), the new model should be inserted at runtime and running instances should be migrated. This feature is not usually implemented in commercial WfMS [Adams, 2007];

2. Difficulties applying ad hoc changes to cope with very small model variations [Rinderle, 2004b; van der Aalst and Basten, 2002; Faustmann, 2000; Jorgensen, 2001] while preserving model consistency. If a particular process requires some special handling (e.g., a special costumer wants to place an order even though he does not have credit), the user should be able to bypass the model and guarantee organizational goals. The majority of existing systems do not support users changing the model of a particular instance at runtime;

3. The models currently adopted to represent work are inadequate to flexibility support [Dourish et al., 1996]. Models should be constraint based, using general rules instead of specifying every action that should be performed.

Two main research streams can be identified in the systems proposed to overcome the identified inhibitors [Han et al., 1998]: metamodel and open-point. Metamodel approaches take into major consideration the structural and dynamic constraints to model adaptations, while open-point approaches define special points in the workflow model where the adaptation can be made. Metamodel approaches offer higher intervention latitude since they are not limited by special points in the model where the intervention can be made. However, they require model consistency checks, while in the open-point approaches the consistency checks are not necessary due to the restrictions in the allowed interventions. The next sections describe metamodel and open-point approaches in more detail. Since some works that deserve mention are outside this classification, the following section is dedicated to them. Other systems supporting unstructured activities are discussed in the last section.

### 3.2.1. Metamodel approaches

Metamodel approaches are usually referred in the literature as providing dynamic and adaptive WfMS and are actually one of the most important research streams to increase flexibility in WfMS. On the occurrence of exceptions, users should be able to change workflow models at runtime, adapting them to the new situation and migrating running instances to the new model without stopping or breaking the system [Ellis et al., 1995; Reichert and Dadam, 1998; Agostini and De Michelis, 2000a; Casati et al., 1996; Sadiq et al., 2000b; van der Aalst and Basten, 2002; Weske, 2001]. Two types of interventions are identified in the related literature [van der Aalst, 2001; Rinderle et al., 2003; Edmond and Hofstede, 2000]: ad hoc changes and evolutionary changes.

These interventions may be defined as:

> *Ad hoc changes are typically applied to a small set of instances and are a reaction to a particular situation that affects some specific processes.*

And,

> *Evolutionary changes result in a new version of the workflow model and result from changes in the business processes that the organization is required to implement (e.g., reengineering efforts or legislation changes).*

Both ad hoc and evolutionary changes must be executed under the system control to keep correctness, avoiding the insertion of deadlocks, unreachable states or inconsistencies in the data dependency model. These solutions define *a set of change rules enabling automated correctness checks*. Two correctness criteria must be taken into consideration: structural and state related. The former concerns schema changes and assures the new model is consistent (cf. the consistency definition in Section 2.1.1). The state related criterion concern the state of the instances to be migrated and verifies if they can be propagated to the new model.

Rinderle et al. [2004a] assessed the correctness criterion usually found in the literature related with dynamic workflow changes. An extension to this work can be found in [Rinderle, 2004b] where additional proposals are evaluated. After defining the dynamic change as the ultimate goal of any change, they compared different approaches based on the semantics of the metamodel approaches. Metamodels were classified according to the evaluation strategies used to trace instance execution during runtime as True-semantics and True/False-semantics. True-Semantics use a marking mechanism to represent future activities, have a true semantics and include a representation formalism such as, for instance, Petri nets. True/False-Semantics use false-tokens to represent skipped execution branches, have a true/false semantics and are based on a graph representation of the workflow model. A brief overview of these approaches is presented below to understand the strengths and weaknesses of the different solutions.

Before initiating the overview, the *dynamic change bug* should be introduced because it is a fundamental restriction to dynamic workflow changes. Firstly introduced by Ellis et al. [1995], the dynamic change bug refers to situations where it is not possible to transfer the instances from the old model to the new one. For instance, in the example of Figure 3.1 the execution order of tasks $T_1$ and $T_2$ is switched[5]. The instance shown in the figure has already executed task $T_1$ but have not yet executed task $T_2$. This instance cannot be migrated to the new model because there is no place where the instance can be located. If the instance is located at place $P_1$, task $T_1$ is executed twice, and if the instance is located at place $P_2$, task $T_2$ is never executed. Another common occurrence of the dynamic change bug is transforming the parallel execution of two tasks ($T_1$ and $T_2$) in a sequence ($T_1$ followed by $T_2$). Since tasks are in parallel, there will be instances that have already finished task $T_2$ and not $T_1$. These instances cannot be migrated because they cannot be placed in any state of the new model: if they were placed after $T_2$, $T_1$ would not be executed, and if they were placed before $T_1$, $T_2$ would be executed twice.

[5] Cf. Appendix A for an overview to modelling workflows using Petri Nets.

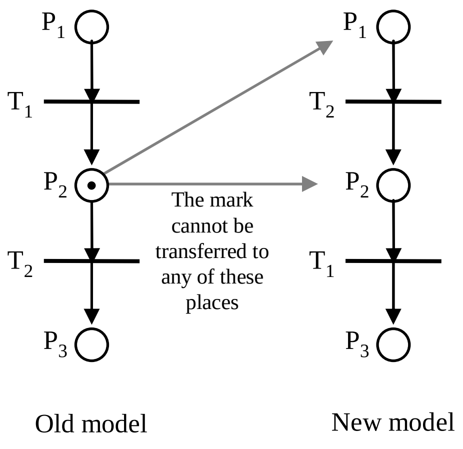


Figure 3.1. Dynamic change bug for switching the execution order of the tasks

One of the first approaches using True-Tokens was presented by Ellis et al. [1995] and later refined [Ellis and Keddara, 2000]. The change region is defined as the part of the net containing all activities affected by the change, the old change region as the sub-net containing all the affected activities in the old net and the new change region as the substitution of the old change region in the new net. A special change class is defined, the *synthetic cut over*, where both the old and the new change regions are maintained in the model. Users identify which states in the old net can be migrated to states in the new net, and define flow jumpers as transitions from the old net to the new. Flow jumpers are special classes of Petri Net transitions and allow the migration of markers in the old net to markers in the new net. When a flow jumper fires, an instance is migrated from the old model to the new one. Instances that are in states that do not fire any flow jumper must wait until they reach one (delayed transfer). For this reason both nets are active during the transfer and until all instances are transferred. Identifying flow jumpers and change regions is done by humans which can be a time consuming and error prone task.

Aalst and Basten [2002] proposed a True-Semantics approach based on a graph equivalence notion established by branching bisimilarity. This concept is formally defined using Petri Net theory, and informally states that two workflow nets are equivalent if one can simulate any behaviour of the other after executing any number of silent actions. Using this notion, the authors present the set of permissible transformations on a given workflow net. Using simple rules, marks in the old net are also transferred to the new net. However, the solution does not handle the “dynamic change bug”, in particular it does not provide a transformation to switch the execution order of tasks [Rinderle et al., 2004a].

Another True-Semantics proposal based on model equivalence was proposed by Agostini and De Michelis [Agostini and De Michelis, 2000a; Agostini and De Michelis, 2000b]. This approach restricts change to free-choice[6] and acyclic[7] elementary net systems to reduce the complexity required to analyse workflow modifications. The authors claim that these nets enable the computation of its main properties in polynomial time [Agostini and De Michelis, 2000a], e.g., the computation and classification of forward and backward jumps linking the model states. (A forward jump allows users to jump to a preceding task in the workflow model and a backward jump to a preceding task. These operations are recognized by many authors as important operations to exception handling [Agostini and De Michelis, 2000b; Reichert et al., 2003; Hsu and Kleissner, 1996].) Even further, it becomes easy to identify the instances that can be migrated to the new model and the tasks that have to wait until they reach a state where migration can be

[6] Free-choice nets are formally defined in Section 3.4. when Petri Nets formalism is introduced, but we will informally introduce them here to facilitate the reading. In free-choice nets the transitions either do not share any input task, or if they share at least one, they must share all. They are a special kind of Petri Nets that can transcribe the workflow model languages that abstract from states between tasks (states are not explicitly represented). If the states are not represented each conditional transition is represented inside the tasks and all transitions share the same input places.

[7] Acyclic nets do not have cycles.

accomplished. However, some authors criticize this approach because of the acyclic restriction [Rinderle et al., 2003].

The approaches described above and proposed by Ellis et al. in [1995], Aalst and Basten [2002], and Agostini and de Michelis [Agostini and De Michelis, 2000a; Agostini and De Michelis, 2000b] basically abstract data dependencies among activities. This is an important issue that will be discussed in Section 3.4.

The WIDE system developed by Casati et al. [1996] uses graph representations of workflow models where iterative loops are allowed and is characterized by True/False-Semantics. Data variables used by the workflow are also represented in the model. The system allows the users changing the model based on a set of schema evolution primitives that guarantee model consistency. The instances for which the new model can replicate the history of the already executed tasks can be automatically migrated. On the other hand, if the history cannot be replicated by the new model, one of the following strategies is possible: keep execution under the old model; abort; rollback the history part that is not replicated with the new model and proceed execution under the new model control; or migrate using ad hoc schemata. When using ad hoc schemata, users define hybrid models that apply to the instances that have equivalent history.

The ADEPT system was introduced by Reichart and Dadam [1998] and uses structured models and is based on a True/False-Semantics. Structured models are based on a set of elementary control structures (sequence, splits and joins) that can be nested, but not overlapped, in the same way as structure programming languages [Kiepuszewski et al., 2000]. The authors chose this model restriction because they claim it is more important to simplify the assessment of consistency, reachability and change realization than having a rich modelling capacity but where the associated analysis techniques are very complex. Even though they selected a different approach, the fundamental assumption is the same as Agostini and De

Michelis. Model consistency, as defined in Section 2.1.1, is assured if the existing loops terminate. The adopted modelling language enables the definition of data constraints between tasks where precedence relations between data providers and data consumers may be established. A marking mechanism on the tasks affords identifying the executed tasks and enables instance history reproduction. Due to this marking mechanism, this modelling language is characterised as having a true/false semantics. Their work is also based on a sound theoretical ground.

A set of change operations have been defined for the ADEPT model, as well as the conditions that the instances must fulfil in order to be compliant with the change. The compliance criterion is further discussed in [Rinderle et al., 2003], where the reduced execution history of the instance is used to verify the executed tasks. This reduced execution history is obtained from the total execution history by subtracting the loop iterations, which is very important if the change is to be applied inside the loop. In a later work, Reichert et al. [2003] discuss the implementation of forward and backward jumps at runtime.

The system purposed by Sadiq [Sadiq, 2000a; Sadiq et al., 2000b] is also a True/False-Semantics approach and enables users to implement the required changes without restrictions at the first approach. Consistency is verified when all changes are implemented in the model and before instances are migrated. When structural consistency is guaranteed, instances are migrated by groups according to their compliance with the change. The approach follows a strict three-phase modification procedure of defining the change, conforming and enact.

Another work in the True/False-Semantics area is purposed by Weske [2001]. Model consistency is formally defined. Then, the instance conformity with the model enables the verification of change consistency. The adopted modelling formalism requires acyclic graphs.

These approaches enable the adjustment of the business process to the actual business process requirements. As systemic approaches increase robustness and maintain control on the system, they rely on the involved actors to perform runtime changes to business processes. They also rely on the modelling formalisms to maintain consistency. We believe that these approaches are adjusted to planned effective unexpected exceptions (cf. Section 2.4 and Figure 2.3) where the user can plan in advance the new model (or even some minor changes) to overcome exceptions.

### 3.2.2. Open-point approaches

Open-point approaches adopt fix points in the model where adaptations may be performed. These approaches have the disadvantage that allowed interventions are not complete enough for some situations that require structural changes [Han et al., 1998]. However, since the interventions are local to a defined point in the model, correctness issues are easy to validate.

One of the first open-point approaches was developed by Deiters and Gruhn [1994]. In this work, the authors describe the Melmac system where composite tasks may have an attribute that enables open-point interventions. These composite tasks can then be replaced during the enactment phase by other composite tasks. Composite tasks are tasks composed by elementary tasks or by other composite tasks. The Mobile system proposed by Jablonski and Bussler [1996] enables the insertion of composite tasks or the replacement of existing ones in the workflow model.

In ObjectFlow, proposed by Hsu and Kleissner [1996], a richer set of operations are available to the user when an exception is detected. After finishing a task, the user may jump forward or backwards in the flow, jump to a different location to implement some special exception handling procedure or insert a new node composed by a set of tasks that should be executed. Some operations are available asynchronously to react to external events, e.g., abort all tasks and jump to a different location to implement some special predefined tasks.

Recognizing the limitation of having fixed points where the interventions can be made, the research by Han [1997] proposes generic open interfaces inside each workflow task where dynamic changes may be made. A predefined set of sub-models are candidates for a specific task and the user selects the ones s/he wants to instantiate for a given case.

A similar approach is proposed by Faustmann [2000] in WAM, where the basic workflow enactment cycle can be broken down to start the adaptation phase. The user responsible for the task can then start a dialog with another user to negotiate a delegation process. This second user, after agreeing on the task details, starts a sub-model to execute the details of the task assigned to him. The model allows replanning, redoing and extending the work process. After the second user receives the task, s/he can replan some of the activities needed to fulfil the task and/or extend the task with some other activities. After the task is finished, the second user informs the first one. At this stage he can inform that the task should be redone, and the responsible decides if that is the case, or if it should execute the task in a different way. This negotiation between the responsible and the second user is in the same direction as the action workflow approach proposed by Medina-mora et al. [1992] described in Section 3.2.4.

### 3.2.3. Other humanistic approaches

Dourish et al. [1996] proposed Freeflow, a constraint based modelling system aimed to decouple the enactment of task execution from the model. The idea is to increase flexibility choosing the next action, since models do not prescribe the next activity to be executed but only constraints between tasks (e.g., task B can not start before task A finishes.) The objectives of this approach are: “1) separate the temporal relationship between tasks, existing in traditional models from dependency relation between activities; 2) users can engage in the component task of a process as appropriate to their circumstances; 3) constraint maintenance is an active on-going process.” The user can execute actions whose constraints are not yet true, thus violating the constraint(s). However, the system is aware of this situation and the started activity is not allowed to enter the complete state until the previous activity is finished. Even though this system augments flexibility, it does not allow the richness of the interventions available on the described metamodel approaches.

The system developed by Borgida and Murata [1999] offers one uniform way of handling deviations in the format of data interfaces, on the data being manipulated and on the workflow model. The authors use concepts imported from object-oriented programming languages, namely the exception handling mechanism and the class hierarchy. Workflows are defined as a hierarchy of classes where each activity is represented by an object with next states defined in the attributes. Special activities represent And-splits/joins and Or-splits/joins. Exception handlers are also workflows. The data being manipulated by the workflow is also declared in classes including the elementary data types and the associated constraints. Using the class hierarchy, it is possible to handle every constraint violation by a single super-class. When the constraint is violated, the system interrupts the executing task and control is passed to the super-class. Then, by changing the attributes of the objects associated with the activities, the user can change the next executing task in the model. This change only applies to the affected instance, because it is implemented in the object and not in the class definition. Even though this is a uniform way of handling deviations, it does not handle external asynchronous events. Also, it does not verify the consistency of changes.

In the system proposed by Adams et al. [2006] the model is not entirely defined when the process is instantiated. Using the case context, the system uses an inferring mechanism, Ripple Down Rules, to derive the most adequate worklet from a repertoire for a particular job of an entire process. This worklet, that is a workflow model designed to perform a particular part of a larger process, is then applied to the job within the process. Worklets may even be designed at runtime increasing the system flexibility. In this approaches users are constrained by the original model devised at design time and interventions are restricted to points in the model where interventions are allowed.

### 3.2.4. Systems fully supporting unstructured activities

This section starts by describing the work by Agostini and De Michelis [Agostini and De Michelis, 2000a; Agostini and De Michelis, 2000b] which is one of the few approaches that has the objective of supporting users in a way similar to this thesis approach. As the authors state “systems supporting articulation work must on the one hand, liberate workers as much as possible from the routine articulation work they need for coordinating themselves (script); on the other, help them to become aware of the situation where they are performing and to negotiate new cooperative work arrangements whenever a breakdown occurs (maps). Finally, they need to be open to continuous change in order to support a continuous update of their maps and their scripts.” In the proposed system, users can execute the actions inserted in the scripts during workflow execution but may also initiate a “multimedia conversation” with another user when some exception situation is detected. These two components are fully integrated, allowing a conversation to be started during workflow execution and a workflow enacted during a conversation. The multimedia conversation component supports the authors Situated-Action perspective [Suchman, 1987; Winograd and Flores, 1986; De Michelis and Grasso, 1994]. The system offers an open architecture where the degree of user involvement varies. Different degrees of involvement are supported through email integration, where users may execute workflow actions if they share the computational email model, where messages embed the execution environment and the data necessary to react to them. Users that only have access to common email can still be involved in the cooperative process, since all messages have an initial text explaining the details of the activity. A template inserted in the message may be filed by the user with the information details about the task and all documents attached in the messages. The message is interpreted by a computer that recognizes the effect of the executed action on the workflow state. In the case the user is not able to finish the work, s/he may say in the reply message the reason (in a special field).

The modelling formalism adopted in this system was discussed in Section 3.2.1 and is based on a special kind of elementary net systems, namely free-choice and acyclic, to enable the computation of the main properties in polynomial time. This property allows the usage of all the algebra associated with these nets, keeping its simplicity high to facilitate user interventions. When an exception occurs, user may perform workflow changes or jumps (forward or backward). Only the interventions that do not insert inconsistencies are allowed.

It is important to mention two other approaches introducing a broader perspective over workflow exceptions. Guimarães et al. [1997] proposed an integrated architecture of formal coordinated processes with informal cooperative processes. [Saastamoinen, 1995] presents an approach focussed on organizational semantics. Petri Nets, outside the scope of the WfMS, define the reactions to the various types of exceptions and should be interpreted as a global organizational reaction to exceptions.

Another important research in this area was proposed by Bernstein [2000], who developed a system to support structured and unstructured activities. The process spectrum from unstructured to completely structured activities was divided into four sub-spectra: providing context; monitoring constraints; planning options based on constraints; and guiding through scripts. The WfMS supports activities in each of these sub-spectra, where the importance of maps decreases from the unstructured to structured activities, while the importance of scripts increase in the same direction. Processes may be transferred between sub-spectra by increasing or decreasing their specificity level.

In the *providing context* sub-spectra, the major goal is to provide context to the users, which can be achieved in the form of check lists, to-do lists and other documents. The system also integrates with email, online group discussions and synchronous communication tools. The information processed in this sub-spectra is not managed by the system. When the user inserts any machine-readable constraint (e.g., a deadline for a task in a to-do list), the system increases the specificity level to *monitoring constraint*, where it monitors every inserted constraint and alerts users when they change state (e.g., the deadline expired). To enter the *planning options based on constraints*, the user must insert a goal or a post-condition in a to-do task. The system then proposes a series of possible approaches using a planner to complete the task based on the goals and post-conditions and on the constraints inserted in the previous sub-spectra. The planner searches in a repository of possible actions and proposes to the user a list of the actions that comply with the restrictions. Then, the user may choose from that list or insert a new action. When the user chooses one action, the system enters the *guiding through scripts* level, where a typical model guided workflow is followed.

The ActionWorkflow™[8] was proposed by Medina-Mora et al. [1992] and is based on the *language action* theory. The theory, developed by Winograd and Flores

[8] ActionWorkflow is a trademark by Action Technologies, Inc.

[1986], intends to structure coordination within organizations as communication acts. The theory has its roots in the speech act theory, which views communication acts as a form of social interaction oriented towards establishing commitments between people aimed at transforming the environment. After establishing this main objective, the authors classify language acts according to a taxonomy that supports a mutual understanding of commitments. This taxonomy enabled the development of The Coordinator whose objective, in Winograd's words, "is to enable a structure of interactions that is effective for coordination within an organization. It uses a formal structure in which regular patterns of language acts are associated with the requests, commitments and declarations of completion. It is based on the fact that these elements are implicit in all interactions where actions are coordinated by people, whether or not they are stated explicitly" [Winograd, 1994]. ActionWorkflow™ uses The Coordinator to establish agreements between performers and clients to implement a desired action. The objectives are negotiated and agreed between the parties and action proceeds until the client accepts the result (atomic loop of action). If the performer has to involve more actors in the action s/he then starts a negotiation with the performer to establish the commitment. One complete organizational process in ActionWorkflow™ is an interweaving of these action loops. The communication acts between performer and client are supported by The Coordinator.

Suchman [1993] criticizes the language action perspective, saying that it does not account for the "irreducibility interactional structuring of talk." By structuring language, The Coordinator acts as disciplining actors rather than supporting their coordination, and plays the role of accountability for human interactions. In a recent survey of this perspective, Weigand [2006] recognizes that communication is much more complex than originally suggested by the speech act theory. Communication is highly dependent on the context; the same communication can relate to several speech acts; maintain an intentional ambiguity; and finally, communication is not something that occurs within an organization but is intertwined with it, since organizations also emerge from communication. Winograd also recognizes that the perspective is not in the main stream of information systems development, as some authors predicted [Winograd, 2006], even though he expects the importance to grow with the increasing power to build large scale information systems. He also recognizes the theory is a “partial account of the reality” because any designed system cannot embody all aspects of human needs and the contexts where the theory is effective to support the design of cooperative systems deserve an intensive study. The assumed bias resulting from the speech act theory must be carefully understood.

## 3.3 Discussion

From the above discussion it is important to realise that systemic approaches are important to increase the WfMS robustness regarding failures and expected exceptions. Robustness concerns handling expected exceptions, using special modelling constructs to automatically deal with these events. Systemic approaches also increase flexibility when dealing with extended expected exceptions (cf. Section 2.4), because they allow user involvement in implementing minor model adjustments to the available handling procedures for the corresponding expected exception. The objective underlying these approaches is to keep the flow of work under the control of the WfMS without any or with minor human intervention. However, systemic approaches propagate failures and expected exceptions to unexpected exceptions whenever they cannot handle them.

When unexpected exceptions are raised, systemic approaches do not provide the required flexibility. In these scenarios, it is necessary to augment the flexibility at runtime by supporting the user performing ad hoc interventions or evolutionary model changes (cf. Section 3.2.1). The majority of authors in the field recognise the importance of integrating mechanisms manually controlled by the users (e.g. [Eder and Liebhart, 1998; Klein and Dellarocas, 2000; Ellis and Nutt, 1993; van der Aalst et al., 1999; Reichert and Dadam, 1998]) or explicitly state that in many situations the role of humans is crucial to collect process specific data not available to the workflow system [Casati, 1998; Luo et al., 2002; Heinl, 1998]. In the approach developed in this thesis, it is assumed that appropriate systemic approaches are implemented by the WfMS. Therefore, whenever an exception reaches the attention of an operator it is unexpected and should be handled by humans. This regards robustness as a primary mechanism and flexibility as the second line of defence to increase resilience.

On the systems aiming to augment flexibility, the metamodel approaches rely on adopting modelling formalisms to support operators implementing changes to the model and migrating the running instances to the new model. Ad hoc changes are distinguished from evolutionary changes. The former case regards adaptations of a delimited set of instances to react to a specific event, while the later relates to generation of new models reflecting some changing environment reality that will prevail. The system ability to effectively apply changes, either ad hoc or evolutionary, is strongly dependent on the modelling formalisms adopted, as discussed throughout Section 3.4.

Metamodel approaches are crucial to support unstructured activities because they support users reengineering business processes at runtime. Most importantly, they can verify if the reengineering keeps system consistency. It should nevertheless be recognized systemic and metamodel approaches have reached a deadlock, imposed by the consistency requirement. It seems that researchers in the field have already achieved results that proof in what conditions the transformations may be valid and in what conditions they are not valid. The lack of publications in the area seems to fundament this statement. Recent publications [Russel et al., 2006; Adams et al., 2006] still cite the article by Rinderle et al. [2004a] when metamodel approaches are discussed.

The open-point approaches described in Section 3.2.2, are an important development towards flexibility because they support direct interventions in the workflow engine, disregarding the need to implement consistency checks in the new model. Instance migration is also easier to implement. Still, their flexibility is limited when higher latitude of interventions is required as they do not go behind the restrictions imposed by the consistency requirement. Therefore, they are not suited to fully cope with unexpected exceptions. Section 3.2.3 discussed several approaches based on a completely different paradigm which disregards the consistency requirement. These solutions are important to enlarge the researcher view on this subject. However, none of them resulted in a developed system that supports unstructured activities with the required flexibility.

The solutions found in the literature handling unstructured activities were grouped in Section 3.2.4. We will distinguish two of them as they were inspiring for the present work. The system proposed by Agostini and De Michelis [Agostini and De Michelis, 2000a; Agostini and De Michelis, 2000b] offers a communication tool supporting a varying degree of user involvement. Users may dynamically get involved in an exception handling and requested to implement workflow actions when necessary. The workflow model may be adapted and users may also jump forward to a task that appears later in the model, or backwards to a task already executed. We believe that the system has relevant features for an effective support to unstructured activities. However, the collaborative diagnosis is not foreseen, neither mechanisms to collect information necessary to decision making. Also, the completeness requirement is not contemplated since users may only implement changes that do not insert inconsistencies. The system proposed by Bernstein

[2000] is also relevant as user support is foreseen for different contexts in the process spectrum. Our approach supports the two limits of the process spectrum: the providing context in Bernstein approach is equivalent to our unstructured support, and guiding through scripts equivalent to our structured support mode. However, the features supported on the other two modes, the monitoring constraints and the planning options based on constraints, may be implemented by our system during the unstructured activities mode of operation. For example, implementing an alarm based on an identified constraint may be done by inserting a monitoring task in our approach. As mentioned before, we have also focused on providing group collaboration to achieve context awareness and decide on the best activity to achieve the desired goals.

Figure 3.2 positions the systemic approaches, metamodel and open-point approaches according to the type of control and planning capacity. The arrow bellow indicates the direction for increasing flexibility and characterizes how the approaches are positioned within the same control type. Three classes were identified in this dimension: systemic, restricted humanistic and unrestricted humanistic. Systems designed to handle failures and expected exceptions have systemic control and planed reaction. In fact, the reaction to these events is pre-planned. Expected exception handling offer higher flexibility because it is easier to plug-in and change the pre-planned reaction to events. Humanistic approaches increase the operators' latitude of intervention and may be applied at runtime, increasing the flexibility to react to unforeseen events. Humanistic approaches were split in the figure into restricted humanistic and unrestricted humanistic. Open point and metamodel approaches are able to support users on both ad hoc and planned interventions. However, they are not able to support unstructured activities because of their limited latitude for interventions (restricted by model consistency).

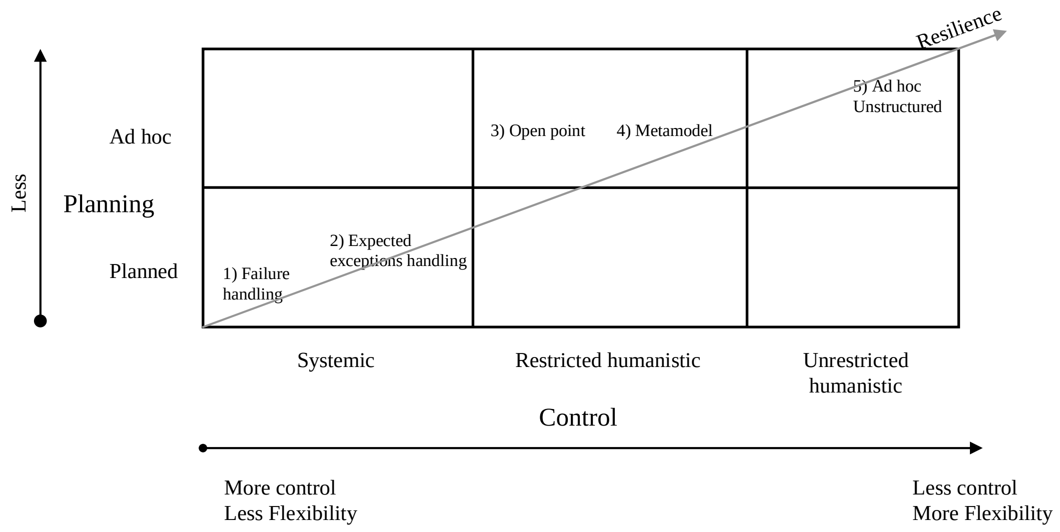


Figure 3.2. Approaches classification according to the control type and planning capacity

The characteristics of our system place it on the top right of the figure and is identified as *ad hoc unstructured*. Here, flexibility is at its maximum degree and interventions are fully ad hoc since the planning capacity is very low. They are also unrestricted by model consistency.

From the figure, we may realize that systemic approaches rely on the stage 1) and 2) on the resilience axes and they only provide systemic support. Metamodel and open-point are at stage 3) and 4) since they do not provide unrestricted support to unstructured activities. The only system that integrates the various operating modes is the one proposed by Bernstein [2000]. However, this system does not account for integrating metamodel approaches, collaboration support or monitoring actions, as mentioned above. We believe that this system is developed in the direction of supporting unstructured activities but lacks some important features.

## 3.4 The expressiveness of metamodel formalisms and their impact in workflow changes

Section 3.4.1 discusses the impacts of metamodel assumptions on workflow dynamic change. The discussion is then used to fundament the metamodel assumptions adopted in this work. In Section 3.4.2, the available operations to implement dynamic workflow changes within the adopted assumptions are discussed.

### 3.4.1. Modelling languages and their impact in workflow changes

Different modelling languages are used in workflow systems. A good survey on this issue, discussing the expressive power of the different modelling formalisms, can be found in Kiepuszewski et al. [2001]. In that survey, existing systems are mapped to Petri Nets and then analysed and compared considering their expressiveness capabilities and evaluation strategy. The mapping of modelling languages to a unique modelling formalism enables a unified view.

Petri Nets, and especially Information Control Nets, a particular type of Petri Nets, have been one of the first modelling formalisms proposed for Office Information Systems [Ellis, 1979]. Since then, the use of classical Petri Nets has been widely adopted by the research community (cf. Appendix A for an introduction to modelling workflows using Petri Nets). Some important Petri Net characteristics justify this wide acceptance [van der Aalst, 1998]:

- Formal semantics – workflows are clearly and unambiguously defined;
- Graphical nature – the graphical presentation facilitates interpretation;

- Expressiveness – they model all the required primitives of workflow models;
- Properties – based on mathematical studies, a wide theory about Petri Nets have been developed that derives important properties;
- Analysis – various analysis techniques are available to evaluate concrete model properties (e.g., consistency and performance factors);
- Vendor independent – this modelling formalism is independent from any vendor and therefore not biased by any implementation specificity.

In this thesis, whenever a modelling formalism is needed, Petri Nets are use. The above mentioned advantages are some of the reasons behind this selection. Nevertheless, a discussion about classical Petri Nets being not capable of modelling data dependencies among tasks is necessary since it is an important issue on workflow management.

Data flow among activities and time related issues can not be simulated using classical Petri Nets. High level Petri Nets, such as coloured or timed, should be used when these facets have strong impact on the analysis. Coloured Petri Nets provide the means to simulate data flow (cf. to [Han, 1997] for a discussion on this issue), while timed Petri nets have been used to verify time related issues associated to dynamic changes [Ellis et al., 1998]. However, since the complexity of the algebra associated with these nets increases significantly, they are avoided whenever possible. ´The adoption of classical Petri Nets, abstracting data dependencies among activities, seems to have gained some momentum in the nineties by the research community [van der Aalst, 1998; Agostini and De Michelis, 2000a; Kiepuszewski et al., 2000; Ellis et al., 1995; Saastamoinen, 1995].

Reichert and Dadam [1998] criticize this approach because they claim it is critical to model data dependencies when implementing workflow changes to avoid losing updated data. Neglecting data dependencies can lead to workflow inconsistencies when applying ad hoc changes at runtime. In their proposed system, data dependencies are simulated among tasks. The increased analysing complexity is compensated by adopting structured workflow models, as mentioned in Section 3.2.1. Structured models can model a subset of the workflows that Petri Nets can model and therefore have less modelling capability.

In conclusion, since coloured Petri Nets have complicated algebra, authors using Petri Nets abstract from data dependencies to decrease complexity. On the other hand, authors taking into account data dependencies restrict their modelling capability to decrease complexity. It thus seems there is a trade-off between model expressiveness and analysis capability. For instance, Agostini and DeMichelis [Agostini and De Michelis, 2000a] (cf. Section 3.2.1) restrict the modelling expressiveness of their system to improve analysing capability. Since our main objective is to have higher latitude for human interventions our adopted modelling approach also neglects data dependencies. With a richer modelling capability, users will have higher latitude for the intervention without inserting inconsistencies. This is important because it will be easier to bring the system back into model control once the exception handling process is finished.

Even though these considerations are important to understand how the metamodel assumptions impact the system capacity to support unstructured activities, they are not the core of this thesis. It should be emphasised that the proposed approach could be implemented on top of any of the described systems supporting metamodel approaches, or even open-point. However, the result would be a system with lower latitude for consistent changes. The trade off between model expressiveness and analysis capability also reflects on the system ability to identify inconsistencies and to suggest recovery procedures. I.e., increasing the model expressiveness also increases the complexity associated to detecting inconsistencies and suggesting recovery procedures. Therefore, some considerations about the relation between the metamodel assumptions and the computing support should also be discussed.

According to Esparza and Nielsen [1994], even for classical nets, the derivation of the main properties can become unreachable in polynomial time. Some restrictions are usually made to maintain the required computing capacity within reachable limits. In [van der Aalst, 2000], Aalst discusses the usage of Free-choice and Well-structured nets. Free-choice nets are nets that, whenever an arc connects a place *p* to a transition *t*, either *t* is the only output of *p* or *p* is the only input of *t* (definition obtained from [Nielsen et al., 1992]). Free-choice nets can model the majority of the systems existing in the market because they usually abstract from states between tasks. The main properties of these nets, and in particular the ones that are most important to this thesis (e.g., consistency and reachable states), can be derived in polynomial time [Esparza and Nielsen, 1994; van der Aalst, 2000]. Well-structured nets balance AND-Splits with AND-Joins and OR-Splits with OR-Joins meaning that they can not overlap. The author claims that “good” workflows can be obtained using this restriction. With this type of nets, consistency and reachable states can also be determined in polynomial time. Given an arbitrary net, to decide if it is well-structured can also be decided in polynomial time. Another type described in Aalst’s work, named S-Coverable nets, seems to be a basic type for any Petri Net that simulates a workflow model. According to the author, it seems that there is a strong correlation between S-Coverable nets and consistency, and that S-Coverability is one of the basic requirements of any workflow net. The consistent nets designed without being S-Coverable should be avoided. Unfortunately, it is not possible to derive the properties of these nets in polynomial time.

### 3.4.2. Dynamic changes in workflow nets modelled using Petri Nets

Once the underlying metamodel assumptions are established, it is important to understand how workflow changes can be accomplished. These operations are important to support humans on their unstructured activities, when they decide to implement recovery procedures over the affected instances.

To perform a set of changes to the model, following the inheritance-preserving transformation rules derived by Aalst and Basten in [2002], in order to maintain model consistency we should consider:

1. Rule 1 – the protocol/projection inheritance (PPS) – insert a consistent net that leaves from a place and reaches the same place;
2. Rule 2 – the protocol inheritance (PTS) – inserts alternative branches of behaviour, i.e., it enables the insertion of a parallel thread in the model;
3. Rule 3 – projection inheritance (PJS) – inserts an entire net between a transition and a place;
4. Rule 4 – projection inheritance (PJ3S) – inserts a parallel branch of behaviour.

The conditions under which these transformations are valid should be consulted in the cited paper. It should be noticed that these transformation rules assure that both nets are consistent, the initial and the transformed one, and therefore the transformation can be done in both directions. For example, for Rule 1 the user can either insert or remove a consistent net.

On the other hand, these transformation rules only consider the structure of the model. Not all the above rules assure that transferring an instance to the new model results in a valid instance. Instance transfer rules were also derived for the direct order of the transformation rule and for the inverse order. Instance transfer from the original net to the transformed one is governed by the rules:

1. Rule $r_{PPS}$ – the transformation Rule 1 only adds alternative behaviour to the original net. Therefore the instances can always be transferred to the new model;

2. Rule $r_{PTS}$ – the transformation Rule 2 only adds alternative behaviour to the original net by inserting another alternative branch. Hence, all the instances can be migrated;

3. Rule $r_{PTS}$ – the transformation Rule 3 only adds alternative behaviour to the original net by inserting a subnet between a transition and a place. Hence, all the instances can be migrated;

4. Rule $r_{PJ3S}$ – when the Rule 4 is used, some checks have to be carried out. If the branch parallel to the inserted one is marked by the instance the new branch must also be marked. The place where the marked is inserted in the new branch is decided by the user (it can be any of the reachable markings of the inserted net);

When the transformation is on the opposite direction of the transformation rule, the inverse instance migration rules are derived:

5. Rule $r^{-1}_{PPS}$ – corresponds to removing a net that leaves a place and terminates in the same place (transformation Rule 1 above is used on the opposite direction). The marks that are in the unchanged net stay in the same place. The marks that are in the removed net are transferred to the place where the removed net was connected;

6. Rule $r^{-1}_{PTS}$ – corresponds to removing an alternative branch of behaviour from a net (transformation Rule 2 above is used on the opposite direction). Again, the mark stays in the same place if it is in the unchanged net. If it is on the removed alternative branch, the user decides whether to transfer it to the initial place of the alternative branch or to the end;

7. Rule $r^{-1}_{PJS}$ – corresponds to removing a subnet from the original net (transformation Rule 3 above is used on the opposite direction). The mark stays in the same place if it is in the unchanged net. It is inserted on the place located after the removed subnet if it is inside the removed subnet;

8. Rule $r^{-1}_{PJ3S}$ – corresponds to removing a parallel branch (transformation Rule 4 above is used on the opposite direction). In this situation the marks inside the parallel branch are removed;

It should be noticed, that jump operations are not studied by this approach. When these operations are used by our solution, the conditions that govern their validity are discussed (cf. Section 5.4).

These transformations were developed for a system with consistency level 4 (cf. Section 3.3). Nevertheless, they are used in our consistency level 5 system that supports unstructured activities because they concern model consistency when replacing the system under model control when the exception handling procedure is finished. They may also be used to inform actors if the transformations they want to implement during unstructured activities are consistent. However, the main focus of this dissertation is supporting unstructured activities. It should be emphasized that any other modelling formalism described in this chapter with the corresponding transformation rules could be used in our approach. We have chosen this approach because it has higher latitude for the interventions even though data dependencies among tasks are not modelled.

## 3.5 Summary

In this chapter, we have defined the resilience property for WfMSs. A resilient WfMS should be robust to resist to failures and expected exceptions, and flexible to handle unexpected exceptions.

Our review of existing systems identified five levels of resilience: 1) systemic approaches to handle failures; 2) systemic approaches to handle expected exceptions; 3) restricted humanistic open-point approaches; 4) restricted humanistic metamodel approaches; and 5) unrestricted humanistic support for unstructured activities.

To support unstructured, activities the system must integrate the five levels of resilience. Integrating the five levels requires the system is able to handle planning and control issues. Planning, when a plan can be issued before handling is initiated, and control when the handling procedure starts before any plan or new model is issued. Few systems integrate these five levels of resilience. The majority of existing systems restrict planning and control by the modelling formalism adopted. Even further, no system supports the collaborative nature, user involvement, monitoring capabilities and decision making involved in level 5.

We have also identified the modelling formalism used in our solution: Petri Nets. We should also stress that our solution may be implemented using any of the existing modelling formalisms since our focus is on supporting unstructured activities. Rules for consistent changes and for migrating running instances were also discussed. These rules are useful when implementing model changes during the exception handling intervention. The system uses these rules to inform users on the consistency of the change and to identify if the system can be placed back into model control.

# Chapter 4

# A Solution to Support the Whole Spectrum of Organizational Activities

This chapter introduces our proposed solution to support the whole spectrum of organizational activities. The system should be able to work under model guidance and adopt map guidance support to unstructured activities when an unexpected exception is detected. Unstructured activities are carried out until the system is back into a coherent state. Then, the user will either place affected instances under model guidance or abort them. The overall system behaviour is modelled by the state diagram in Figure 4.1.

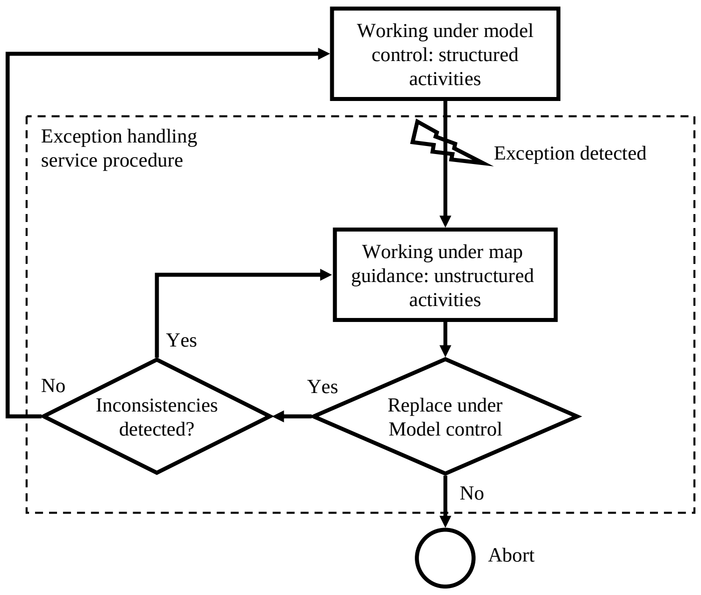

Working under model control: structured activities
Exception handling service procedure
Exception detected
Working under map guidance: unstructured activities
Yes
No
Yes
Inconsistencies detected?
Replace under Model control
No
Abort

Figure 4.1. Solution’s state diagram

Our *proposed solution* refers to the computer base system that supports the whole spectrum of organizational activities including the standard WfMS system and the developed functionality to support unstructured activities. Users are one of the entities within the overall organizational system that interfaces with the proposed solution but are not included.

In our solution, we implement the extended reference (cf. Section 2.1.2). The developed functionality that supports unstructured activities runs on a workflow engine. A dedicated model implements the components to support unstructured activities. When a new exception is detected, the workflow model is instantiated and the system starts supporting unstructured activities. This corresponds to the transition from structured activities to unstructured activities in Figure 4.1. The types of exceptions that trigger this transition have been described in Section 2.4 and classified as ad hoc effective unexpected exceptions. Unstructured activities are carried out until the situation is back in control. When users realize that the situation is resolved, and they want to abort the affected instances nothing else needs to be done. On the other hand, if the instances should be placed again under model control, the system should verify whether inconsistencies were inserted and support users on removing them. When inconsistencies are removed the system is placed back under the WfMS control, i.e., the control is placed back under the WfMS system.

This chapter is dedicated to describe our solution. Hence, in Section 4.1 we present a conceptual approach to the solution. Section 4.2 describes the more detailed solution state diagram addressing the above mentioned behaviour: maintain model-based work whenever possible and change to map guidance whenever the scope is outside the limits under which the work models were designed. The solution state diagram is implemented by the exception handling workflow explained in Section 4.3, where the basic exception handling functions are identified: detection, diagnosis, recovery and monitoring. The main activities carried out by each one of these four functions and their inter-relations, as the handling procedures evolve, are also analysed. The detection mechanisms are discussed in Section 4.4. The following sections are dedicated to discuss the remaining exception handling functions. Section 4.5 explains the exception diagnosis, where a special focus is made on the exception characteristics that may help the users on the selection of the most appropriate actions to carry out during the intervention. Section 4.6 discusses recovery and monitoring functions into the wider perspective of the handling strategies that users may adopt. Both functions represent user’s actions upon the workflow engine even though their objective is different in the sense that recovery actions try to bring the system back to a coherent state while monitoring actions collect information about the situation. The handling strategies adopted during the exception handling procedure are also classified in Section 4.6.

## 4.1 Conceptual approach

Figure 4.2 is our proposal for the extended WFMC reference model[9] (cf. Figure 2.2) that can support the whole spectrum of organizational activities according to the state diagram introduced in the previous section. When the system is supporting structured activities, the traditional WfMS has control over activities and the exception handling service is inactive. When an exception is detected, the exception handling service interrupts the WfMS execution and the system starts supporting unstructured activities. Our solution implements the exception handling service that connects to the traditional WfMS through interfaces A, B and C.

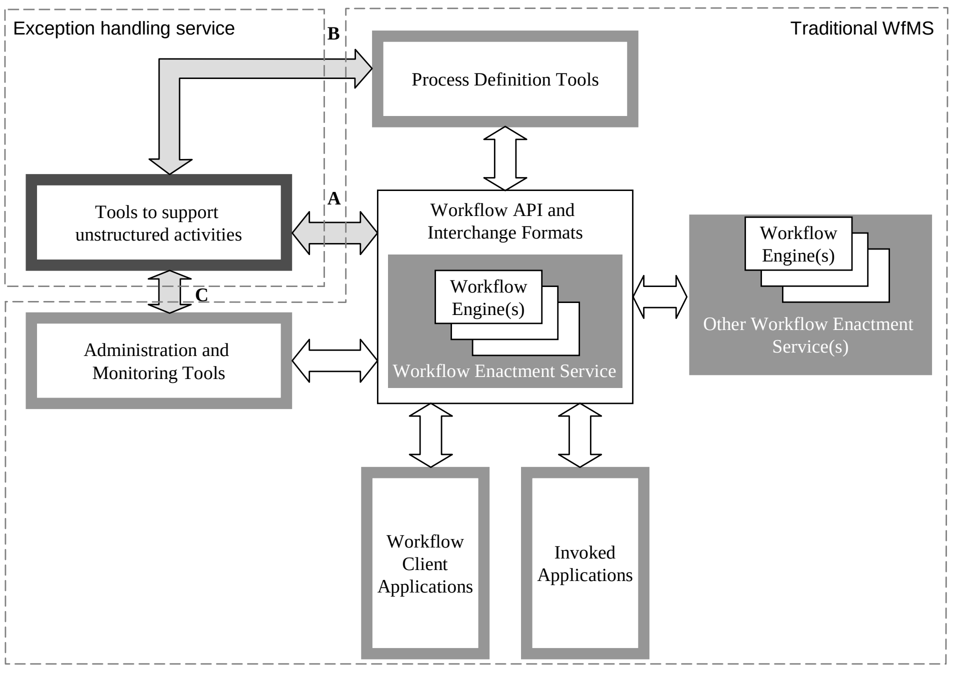

Exception handling service
B
Traditional WfMS
Process Definition Tools
Tools to support unstructured activities
A
Workflow API and Interchange Formats
Workflow Engine(s)
Workflow Enactment Service
Workflow Engine(s)
Other Workflow Enactment Service(s)
C
Administration and Monitoring Tools
Workflow Client Applications
Invoked Applications

Figure 4.2. Extended WFMC's reference model

[9] Figure 2.2 is repeated to simplify the reading

During unstructured activities support, the system supports the following functionality:

(i) Escalation;

(ii) Monitoring;

(iii) Diagnosis;

(iv) Communication;

(v) Collaboration;

(vi) Recovery;

(vii) Coordination;

(viii) Tools to determine the best solution;

(ix) History log.

The relevant organizational actors should be involved in the exception handling activities. Appropriate organizational levels with adequate decision authority should participate in decision making and action implementation. The *escalation* mechanism allows the involvement of organizational members in the process. On the other hand, to support the group of involved users overcoming the exceptional situation the system must:

1. Support users on understanding the situation – diagnosis;
2. Support users on deciding the most adequate actions to overcome the situation – recovery;

To facilitate the *diagnosis*, users should be fed with quality information about the peculiarities of the situation at hand. Since information and knowledge about the event are spread through the organization and the environment, our proposed solution implements *monitoring* mechanisms to collect relevant data and enable knowledge sharing among participants (cf. openness requirement in Section 2.5). This shared effort should be supported by appropriate *collaboration* and *communication* mechanisms that facilitate common situation awareness. A *context mapping* should be provided to users and groups where affected instances and processes are the main focus. This interaction context, as described by Zacarias et al. [2005b], provide a "shared empiric, syntactic, semantic, and pragmatic space for actors" that facilitates knowledge distribution. *Diagnosis* is also supported by a situation description component used to classify the event according to several dimensions that had to be developed by this research as part of the proposed solution. Since the situation may evolve over time, users may change the description as new information is collected that changes users' perception on the situation.

The decision making process on the recovery actions to implement may be characterized as a mutual adjustment coordination mechanism, identified in Section 2.2, when the Organizational Sciences perspective was discussed. The *collaboration* and *communication* mechanisms support the implementation of mutual adjustment. (Mutual adjustment requires combined of communication, coordination and collaboration.) These mechanisms facilitate involving the adequate users on the decision making process.

It should be emphasized that during unstructured activities support, the coordination facet implemented by the WfMS is relaxed since users gain control over orchestrating their activities. This unavoidable characteristic of the solution imposes a special focus: **users should coordinate their activities**.

The functionality *tools to determine the best solution* accounts for application environments where special tools may be used to support both the understanding of the situation and the decision process (e.g., operations research algorithms in a lot manufacturing company may support users on calculating the lots that should be manufactured when reacting to an unexpected change in demand). These tools are highly dependent on the application context but they may be implemented as components of the solution accessible to users during the exception handling process.

Finally, our solution maintains a *history log* for the situation description and for all the implemented activities. When new values are defined for the situation description, the old values are stored in the historical log. This log may be consulted during the event or on future similar events.

It is relevant to discuss where the control over the system lies at the different stages of the handling procedure. The resilience property discussed in the previous chapter assures that the WfMS is characterized by robustness and flexibility. Systemic approaches are designed to improve robustness maintaining *system control* over activities even on the presence of failures and expected exceptions. When systemic approaches are unable to solve the situation, *human control* over activities becomes dominant. Events are therefore handled first at the level where they occur and escalated to a different level whenever required. This assures that events are handed at the adequate level. Figure 4.3 illustrates examples of **organizational trajectories** of exception handling procedures. Bellow the line, control is on the system side and activities proceed according to the model. When the line is transposed, operators obtain control over unstructured activities. These activities are carried out until control may be passed again to the system.

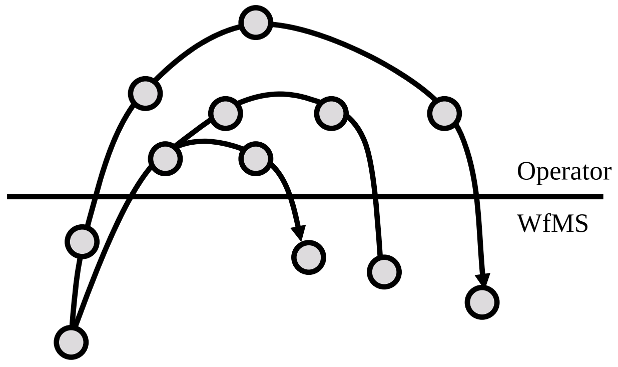


Figure 4.3. Examples of possible organizational trajectories for the exception handling procedure

Usually, the exception is detected by the operator involved in a failing task becomes involved in the handling procedure. Until s/he decides to involve more users in the process, the exception *scope concerns a single operator*. Then, the user may decide to involve more persons and escalate the exception to other colleagues. If users within the same department are involved, the exception *scope is group*. If more than one department is involved the *scope will be organizational*.

In the example of Figure 4.4, the employee that detects the exception escalates the event within his group. Then, the group's manager realizes that employees from another department should be involved and the scope becomes organizational. When other departments finish their collaboration, the exception scope becomes group again. In the example, the group later recognizes the need to involve other departments. This example illustrates the dynamic aspects of group composition during the handling procedure. To assure that the exception has a driver at any point of its handling procedure, at least a responsible person must always be associated. Our solution does not support multiple independent groups handling the event. Everyone involved in the exception handling effort shares the same context and have the same mechanisms to handle the event and to share information.

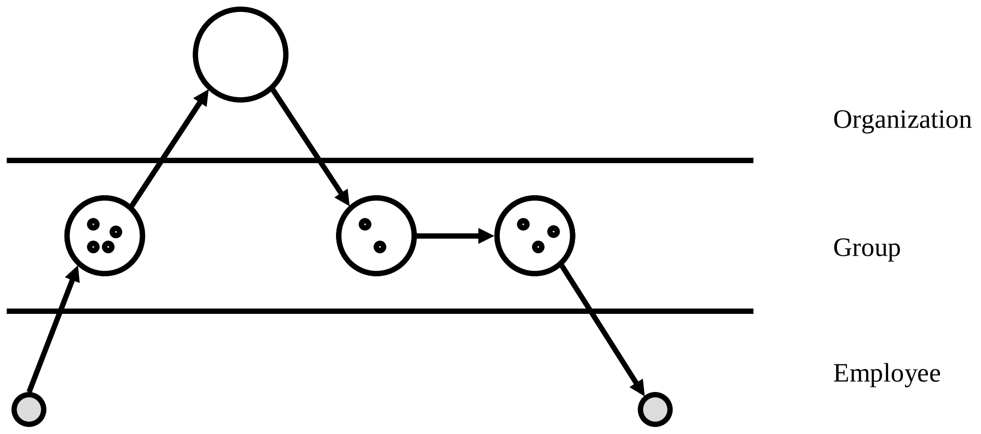


Figure 4.4. Illustration of the organizational escalation of an exception handling procedure

## 4.2 Solution's state diagram

This section starts by introducing the state diagram that governs the system changes from model control to map guidance and then back to model control. The state diagram is then discussed in the light of the motivating example. The necessity of mixing both behaviours is also recognized by Zacarias et al. [2005a]. According to the authors, ad hoc behaviour is adopted to react to exceptions. However, their work aims to define an organizational model that describes the execution of business activities and not the support of business processes using WfMSs.

As mentioned before, to support the whole spectrum of organizational processes, the WfMS should able to switch the operation mode from structured support to unstructured and then back to structured. The system should follow the organizational trajectory and users should be involved in the process when required. Understanding how the system evolves is very important to understand the solution as a whole. The high level state diagram for one affected process instance is depicted in Figure 4.5. If an event affects more than one process instance, it is replicated to multiple state diagrams managed by the system. Normally, a process instance is under model control. Basic failures, application failures or expected exceptions are handled by the WfMS according to consistency levels 1 and 2 (cf. Section 3.3). In these cases, human intervention is kept strictly limited to some concrete exception handling activities that may be applied under model control and do not affect overall system consistency.

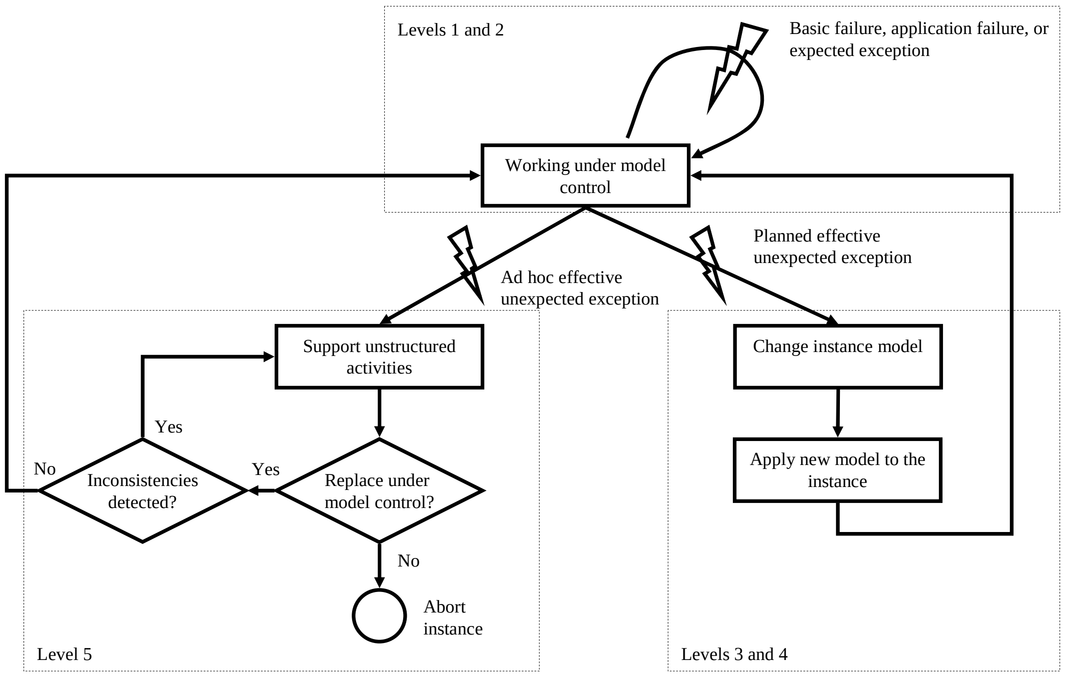

Levels 1 and 2
Basic failure, application failure, or expected exception
Working under model control
Ad hoc effective unexpected exception
Planned effective unexpected exception
Support unstructured activities
Change instance model
Yes
No
Inconsistencies detected?
Yes
Replace under model control?
Apply new model to the instance
No
Abort instance
Level 5
Levels 3 and 4

Figure 4.5. Solution state diagram for one affected instance

Planned effective unexpected exceptions are handled by the WfMS according to consistency levels 3 and 4. The techniques to handle this type of exceptions are not the main focus of our research. We rely on metamodel and open-point approaches (cf. Section 3.2) to handle these scenarios. After the exception is detected, the new model is issued and the instance is migrated. Then, the system may be placed back under model control. In these levels, control is kept in the system and human intervention is the necessary to define ad hoc interventions or new work models.

Finally, systems with consistency level 5 handle ad hoc effective unexpected exception. Since these systems supports unstructured activities carried outside the consistency boundaries, i.e., users may insert inconsistencies in the instances (cf. completeness requirement Section 2.5), when the exception handling is considered finished by the users, they may decide whether the process instance should adopt model control, continue outside model control or be aborted. If model control is the choice, the system will then analyze model inconsistencies and either redeems model control or notifies the users about existing conflicts, while continuing supporting unstructured activities. Model consistency analysis uses metamodel approaches (cf. Section 3.2.1) because they provide higher latitude of intervention than other approaches.

It is important to define how the system reacts if another exception is raised during any of the above interventions. Since there are two exception types in two different states, four scenarios will have to be investigated: two exception types during unstructured activities; and two during new model. Figure 4.6 shows the complete state model with system reaction for the four mentioned situations.

We will start by the two situations depicted on the left: exceptions raised during unstructured activities. When a planned exception is raised, the information should be inserted into the situation description. Affected users should be informed because this new information may condition their recovery procedure (e.g., if a legislation change implies a new instance model for the affected instance, users should be informed because they may have to take this information into consideration during their unstructured activities). Users will also take into account that the instance will be placed under control of the new model if it is to be placed again under model control. When an ad hoc unexpected exception is raised, this new information should be associated to the situation description and used by the involved actors. The old values for the situation description are stored in the history log since they may contain relevant information about the situation evolution.

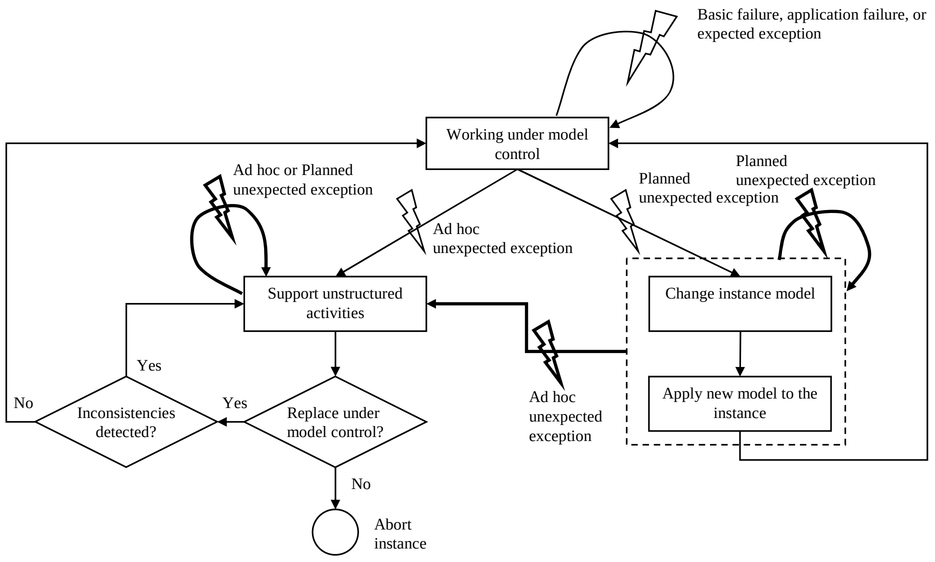

Basic failure, application failure, or expected exception
Working under model control
Planned unexpected exception
Planned unexpected exception
Ad hoc or Planned unexpected exception
Ad hoc unexpected exception
Support unstructured activities
Change instance model
Apply new model to the instance
Ad hoc unexpected exception
Yes
No
Inconsistencies detected?
Yes
Replace under model control?
No
Abort instance

Figure 4.6. Complete solution state diagram for one affected instance

The system reaction to the occurrence of exceptions during model changes is represented by the darker arrows on the right side of the figure. If another planned exception is raised during the handling procedure of a previous planned exception, this new situation must be taken into account during the model design. The final model must account for both exceptions and then the instance should be migrated. On the other hand, if one ad hoc exception is raised, the system changes to support unstructured activities associating to the exception the information that the model must change before the instance is placed back into model control.

Consider, for instance, the 9/11 situation where a plane reports an emergency to the air traffic control operator. This is an expected exception, since operators have standard procedures covering such emergency situations. Usually, the air traffic control operator will try to arrange for the nearest airport to accept the plane. Every other plane on the way is informed and flight plans are redefined if necessary. The airport also starts several standard procedures handling the emergency situation. On this type of situations, the air traffic control operator knows the right people to involve (including authorities, other affected colleagues, etc.), what to do with the affected planes and the type of information that should be sent to the pilots.

However, on the 9/11 event, after the order to land all planes was issued, the Memphis control centre operators scrapped normal air traffic procedures and decided that every controller should follow their assigned planes until landing. Usually the planes are transferred from a proximity operator to an airport operator when they get close to the airport. But since the number of planes to land was very high, they decided that it was more efficient to eliminate these transfers between operators, reducing the synchronization and information overloads. Suddenly, the air traffic controllers started working under a completely new choreography. As reported, all over the country the controllers had to find out the best solution to overcome the problems they faced in their areas to safely land the planes. During this period, the air traffic control system in the US was operating with unstructured activities.

When the situation finally got under control, i.e., officials were convinced that no hijacked planes were in the air, they smoothly started rescheduling and allowing commercial airplanes to take off to their destinations. The system therefore was step by step being lead to model control.

## 4.3 Basic functions

As mentioned before, unexpected exception handling is a problem solving activity that requires understanding the situation and implementing the required activities to overcome the exceptional situation. We distinguish four functions for the problem solving process of unexpected exception[10] handling:

- exception detection
- situation diagnosis
- exception recovery
- monitoring actions

The majority of authors identify the first three functions [Sadiq, 2000c; Dellarocas and Klein, 1998]. However, as it was discussed before (cf. the openness requirement in Section 2.5) and will be further developed bellow, we posit that monitoring actions play a key role in unexpected exception handling.

Since detection is only important for triggering the handling procedure and is independent from the other functions, this section only describes the other three functions. Exception detection is described in Section 4.4. In our solution, we advocate an intertwined play between diagnosis, recovery and monitoring until the exception is resolved. That is to say, the diagnosis is not considered to be complete on the first approach but rather through an iterative process where different actors may collaboratively contribute and information collected from

[10] Remember that we will use unexpected exceptions to refer to ad hoc effective unexpected exceptions whenever it is not necessary to distinguish them.

monitoring and recovery actions is also used to improve it. It should also be stressed that both the exceptional situation and perception of the situation may change along this iterative process, as new information is made available and being processed by humans.

As an example, using the 9/11 case, the already mentioned white board displaying information about the planes suspected to be hijacked was very important to manage the situation and decide the next steps. The white board was constantly being updated and it reached 11 planes (including flight 77, which was indeed hijacked.)

This type of activities, categorized in our solution as monitoring actions, is necessary to control the progress of the whole exception handling process. They allow users to collect up to date information related to running process instances and tasks. Considering again the open nature of the framework, these monitoring actions may also bring environmental information to the system: i.e., the white board was an external component that had to be linked with the system, for a time period, in order to facilitate the exception handling. Other examples of external services would include, for instance, geographical information systems.

After diagnosis, users may carry out recovery actions. The open nature of the solution indicates that the recovery actions do not always run in the inner system context and thus some linking mechanism is necessary to bring environmental information to the system. This issue will be addressed later in more detail.

Figure 4.7 shows the solution handling cycle, illustrating the intertwining play between diagnosis and recovery. The handling activities are initiated by the exception detection. Finally, when the exception is solved the handling activities finish. This cycle corresponds to the unstructured activities support box in the state diagram of Figure 4.5.

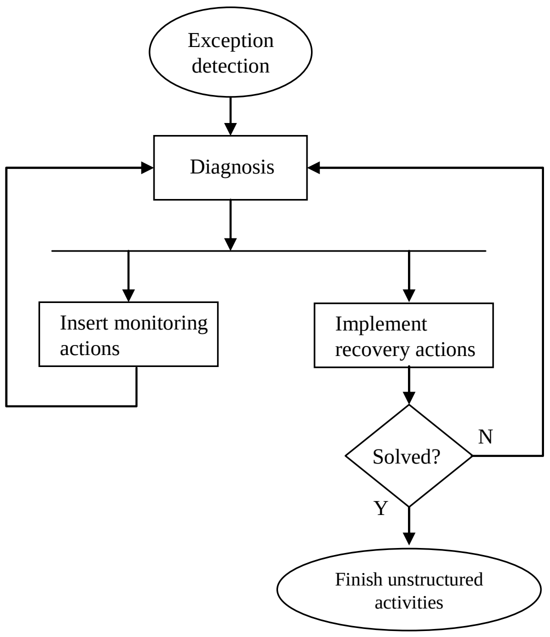

Exception detection
Diagnosis
Insert monitoring actions
Implement recovery actions
Solved?
N
Y
Finish unstructured activities

Figure 4.7. Exception handling cycle

Recovery and monitoring actions may be grouped into a broader concept named *handling strategies*. During unstructured activities, actors use the available information to decide the next steps (that can be information collecting and/or recovery actions). Every implemented action, either monitoring or recovery, may bring new information to this cognitive process that proceeds until the situation is considered solved. Users are supported by these information flows that contribute to situation awareness and implement a feedback process where new information is always added to the system and may be used on the coming decisions.

## 4.4 Exception detection

Exception detection has been extensively studied in previous works [Dayal et al., 1990; Casati, 1998; Chiu et al., 2001; Sadiq, 2000c; Mourão and Antunes, 2004c]. We distinguish between manual and automatic detection because they behave differently from the user's perspective. As will be explained bellow, synchronicity to process execution is also important to understand how the detection mechanisms can be implemented.

The exception classification that classifies exceptions according to the event that generates them is most important for the detection phase. The classes identified in Section 2.3.1 were: workflow, data, temporal and external. We propose two new classes in this classification: non-compliance and system/application. We present a summary of the classes already defined, for the sake of comprehensiveness, and a definition for the new ones:

- *Workflow* – triggered when a task or a process is started or ended and refers to the execution of the workflow itself. These exceptions are synchronous to process execution;

- *Data* – identified within the task that generates an error condition associated to the data. They refer to workflow relevant data and are synchronous to process execution;

- *Temporal* – triggered on the occurrence of a given time stamp. These temporal events may be further divided into: timestamps, periodic and interval. These events are asynchronous to process execution because their firing does not depend on the execution of any workflow activity;

- *External* – activated by external signals and are asynchronous to the workflow execution.

- *Non-compliance events* – triggered whenever the system cannot handle the intended process due to differences between the tasks and the goals modelled by the process; Asynchronous.

- *System/application events* – triggered when the system is not able to recover from lower level failures, such as database, network or application failures (lower level failures are propagated as semantic failures [Eder and Liebhart, 1996]). They are asynchronous to process execution.

When an expected exception that can not be handled by the appropriate procedure is propagated to the unstructured activity support system, that exception is characterised according to the original event, i.e., any of the above mentioned classes.

Classes of the type data, temporal, workflow and system/application are automatically detected by the workflow engine and delivered to the exception handling service (cf. Figure 4.2). The implementation of detection mechanisms will be described in Section 5.3. However, it does not mean that all exceptions in these classes are always automatically detected. They can only be detected if the corresponding exceptional behaviour is predicted in the model or in any special construct supported and implemented by the WfMS. The workflow modeller must always predict the situation before the instance is running. If, for example, an unpredicted infinite loop (workflow exception) is being executed and no special construct is inserted, the exception must be manually triggered. External and non-compliance exceptions are manually triggered by users.

The synchronicity affects the way special constructs can be inserted in the system to automatically detect the exception. If the event is synchronous to process execution, standard modelling constructs can be used. When the event is not synchronous, some special modelling mechanism should be used in the detection. These issues will be discussed in more detail in Section 5.3 in respect with the implementation of the automatic detection mechanisms.

Finally, as mentioned in Section 2.3.1, application data errors are treated as application errors and result in an exception of the system/application type. This enables a coherent treatment of every exception raised by an application. Only the workflow relevant data that did not result in an application error, although triggering a workflow exception condition, results in a data event type. These exceptions occur, for example, due to data type errors or workflow conditions affecting more than one instance that the application does not detect (the same trip being booked by two different instances is a good example.)

## 4.5 Exception diagnosis

A good understanding of the exceptional situation is crucial for users to take the right decisions on which recovery actions to adopt. As already mentioned in this chapter, providing rich context information is critical for convenient map guidance. The information should also support the diagnosis and decision on the best handling strategies. The diagnosis is mostly dependent on a detailed and accurate assessment of the exceptional event.

During the 9/11 example, in the FAA's centre at Herndon, "as Sliney, the operation's manager, moves around the room, a handful of air traffic specialists follow. Together, they have decades of experience and no one hesitates to share an opinion. But without good information, Sliney knows that any decision might be risky. Amid the shouts and chatter and conflicting reports, he reminds himself: Don't jump to conclusions. Sort it out." The claim "not to jump into conclusions, sort it out," is one key aspect in our solution: the recovery actions should be driven by qualified updated information. On the other hand, the observation that the most qualified air traffic controllers are at the room is also very relevant and highlights the fact that the persons involved in the situation is also an important issue.

Using the classifications described in Section 2.3 and some new added characteristics, we propose the following dimensions:

1. *Scope* – **process specific** when only a set of instances is affected; or **cross specific** when various sets of instances are affected. At least one instance must always be associated to the exception;

2. *Detection* – **automatic** if the exception is automatically detected by the system; or **manually** if the exception is manually triggered;

3. *Event type* – refers to the event that generates the exception and can be one of the following types described in Section 4.4: **data**, **temporal, workflow, external events, non-compliance** or **system/application**. The assessment of the event type is mandatory, because it directly impacts the handling phase;

4. *Organizational impact* – **employee**, when only a limited number of employees in the same department are affected by the exception; **group**, when more than one department is affected; and **organizational**, when the overall organization is affected. A responsible person must always be associated to the exception;

5. *Difference to the organizational rules* – **established exceptions** occur when rules exist in the organization to handle the event but the right ones cannot be found; **otherwise exceptions** occur when the organization has rules to handle the normal event but they do not apply completely to the particular case; and **true exceptions** occur when the organization has no rules to handle the event;

6. *Complexity of the solution* – **easy**, when the optimal solution can be easily obtained in an acceptable time; **hard**, when the optimal solution is not obtainable within an acceptable time. In this dimension, complexity is not defined as the overall complexity of the handling procedure, but rather an estimation of the possibility to define a cost function based on the available data. Whenever such a function exists, this dimension provides an estimate of the complexity degree to calculate the optimal solution;

7. *Reaction time* – **quick**, when the reaction to the exception must be as fast as possible; **relaxed**, when the reaction time is not too critical but some decisions must be taken within a time frame imposed by the instance(s); **long**, when the reaction time is not critical. This information is mandatory;

8. *Time frame to achieve solution* – **quick**, when the situation is expected to be resolved in few working units, normally minutes or hours; **relaxed,** when the time frame is more relaxed, although being a parameter to be taken into consideration, normally measured in working days; and **long** when time is not a critical issue.

As detailed bellow, only the *scope*, *organizational impact*, *event type* and *reaction time* dimensions must be set by the detection function. The other dimensions may by set or not by the users, according to their perceptions of the situation. This avoids inserting irrelevant or inadequate information into the system. Note also that the characterization of a specific exception may be redefined by the users whenever more information is collected. The old values are always preserved in a chronological record.

Bringing back to our discussion the 9/11 case, and considering the first exceptional event, the detection was manual and occurred when the controller realized that AA flight 11 stopped answering calls and the transponder signal disappeared from the radar screen. This *process specific* situation affected only one instance. The *time frame to achieve solution* was relaxed, since the controller had to follow the event realizing if it was a serious trouble with the plane or some transitory malfunction. Some other diagnosis information would include: it was a *data event type*; the *organizational impact* affected only one employee and the *difference to the organizational rules* is an expected exception, where the controller knows the right procedure to apply.

The dimensions listed above are the ones that can be classified immediately after the exception is detected. The *complexity of the solution* is therefore undefined, since the controller does not yet know what is going on with the plane. The controller will thus monitor the situation until the context or the perception of the problem changes.

When the controller heard a strange accent in the cockpit saying through an open microphone "we have some planes, Just stay quiet and you will be OK," the situation changed and the exceptional event was escalated to the control centre in Herndon. This is a type of situation to be followed by the central office with high priority: the *organizational impact* changes to include the national operations manager while the *time frame to achieve solution* is maintained in relax mode. The *time frame to achieve solution* may be relaxed, because hijacked planes usually follow some course to an airport and thus do not demand fast recovery, such as an immediate crash emergency.

When the second hijacked plane hit the south tower, the diagnosis changed again. The *time frame to achieve solution* had to change to quick, the *organizational impact* now affected the whole air traffic control organization, the *complexity of the solution* changed to high and the *difference to organizational rules* corresponded to a true exception. As a consequence of the new diagnostic, the national operations manager started wondering how many and what planes were in the hands of the hijackers. He realises that he needs to collect more information, e.g. to identify the affected instances.

The information listed above is a general characterization of the exceptional event. The above information should be complemented with additional data:

1. Affected workflow instance(s) – a list of the affected workflow instances for process specific situations or a rule identifying the set of affected instances for cross specific situations. As mentioned above, this list must have at least one element;

2. Affected task(s) – identification of one or several tasks where the exception was identified. For instance, interval events and workflow events are associated with one specific task while data events may be associated with several tasks;

3. Data structures – associated to *data events* and store information about the data that originated the exception;

4. Data timers – associated to *temporal events*, they store information identifying the expiring timer and the type of exception (i.e., timestamps, periodic or interval);

5. Model deviations – this information applies to *non-compliance events* and identifies a list of tasks that should be inserted, modified or removed.

The following additional parameters enrich the global characterization:

1. A brief textual description of the event. This information applies to external events, since they can be triggered by humans;

2. Root cause – a textual description, produced by a human, with the perceived root cause for the exception.

3. Person responsible – a name of a human that is responsible for the exception (cf. Section 4.1). This name may be either selected by the system from the list of persons associated to affected tasks or produced by a human, as with the root cause mentioned above. The person responsible is a mandatory field meaning that a name from inside the organization must always be associated to the exception.

4. Impact – for every affected instance, the system may also provide information about deadlines and the impact to the organization (based on metrics such as the diversity and number of affected tasks).

The following information is mandatory: event type, at least one affected workflow instance, person responsible and reaction time. The event that triggered the exception belongs to one of the classes mentioned in Section 4.4. During the detection, it is easy to identify the event type and therefore this information should be provided. Since one exception affects at least one running instance, our proposed solution assures that one instance is always associated to the event. As mentioned before (cf. Section 4.1), to assure the exception has always a driver, a responsible person must always be defined. The responsible may change during the handling procedure. Finally, to assure that events requiring quick reactions receive operator’s attention, the reaction time is also mandatory.

## 4.6 Exception handling strategies

The following dimensions to classify exception handling strategies are identified:

(i) *Objective of the intervention* – further division presented below;

(ii) *Communication type* – **synchronous** or **asynchronous**. This dimension classifies the way people exchange information to share the situation knowledge/understanding. This is a common CSCW classification [Ellis and Nutt, 1993] that enables choosing the most adequate tool to support collaboration;

(iii) *Collaboration level* – **one person** solves the situation; several persons solve the situation in a **coordinated mode**; or several persons solve the situation in a **collaborative mode**. It should be emphasized that this dimension is focused on implementing recovery actions. The involved actors may implement recovery actions in a coordinated mode, meaning that they are aware of each other's activities, while in collaboration mode they only know a general description of the intended objective agreed during the last collaborative section;

(iv) *External monitoring* – there is either **enough information** to achieve the best solution or **additional information** must be collected from the environment to support situation diagnosis and decision making. (Gathering information from the system (internal monitoring) may be achieved by inserting tasks and is not regarded as a strategy.);

(v) *Tools to determine the best solution* – either **no external decision aids** are required or there is a need of **advanced support** to achieve the best solution.

This information is associated to every exception raised in the system. It must be emphasized that, likewise diagnosis information, the handling information may change over time as more data about the exception is obtained. A chronological record of the selected values is kept in the system to be consulted by the involved users.

The *objective of the intervention* is related to the high level objective of the exception handling procedure. It is further divided into [Eder and Liebhart, 1996; Agostini et al., 2003; Reichert et al., 2003; Chiu et al., 2001; Sadiq, 2000c]:

- *Abort* – abort the instance. This objective is further divided in: hard or compensate some tasks. In hard abort tasks are terminated with no further action, while in compensate some tasks some activities will be carried out to compensate some of the already executed tasks in the model before instance are terminated;
- *Decrease time* – decrease completion time to meet deadline;
- *Increase time* – increase completion time to release resources;
- *Recover from a system failure* – after a system failure, the objective is to replace the system back in a coherent mode so that the normal flow can proceed under control of the WfMS;
- *Recover from an application failure* – after the failure of a specific task, the objective is to recover the application and place the system back in automatic mode;
- *Lowest penalty* – recover to achieve the lowest penalty possible, i.e., the exception has already impacted negatively the organizational goals, and the objective is to minimize that impact;

- *Delay this task* – this objective can be useful to release some resources necessary to increase the execution time of another process/instance;
- *React to environmental changes* – this normally requires a model change.

This classification affords linking the high-level handling strategies with a specific set of tasks available at the system level. The *communication type* expresses how the collaboration support component will interconnect the persons involved in the handling process. Two types of communication are differentiated: synchronous and asynchronous. In synchronous communication, the involved actors exchange information in real time (in face-to-face interactions or using some means to transfer information), whereas in asynchronous communication information is exchanged in deferred time meaning that information is not received at the same time it is sent.

As mentioned in Section 4.1, coordinating activities among users is an important aspect of our solution because the coordination facet of traditional WfMS is relaxed. During unstructured activities, users have the responsibility of orchestrating their activities. The two modes of operation identified in the *collaboration level* strategy reflect the concern with coordination. In a coordinated mode, users may choose any available tool to coordinate their activities. These tools may be computer supported, such as sending an email or start an instant messaging conversation, or without computer support, such as telephone conversations or face-to-face meetings. If users chose a computer based tool, the solution may support the collaboration effort and it is carried out within the solution's boundary. However, if a non-computer based tool is the choice, the collaboration effort is carried outside the solution's boundary and no information can be automatically collected from this interaction. In a collaborative mode, the coordination aspects are not relevant since users implement their activities in an independent way. They coordinate their efforts only on the next collaborative section.

Still considering the coordinated mode, one has to be aware of concurrent changes made to work models. When ad hoc changes are applied in a coordinated mode, every change is seen as an independent change and the resulting work model results from the composition of previous changes. Therefore, the structural and dynamic checks are made on the instance with respect to this new model. However, in the case of concurrent ad hoc changes carried in a collaborative mode, the work of Rinderle [2004b] must be taken into consideration, because actions carried out by different users without any prior agreement may conflict. In her work, Rinderle discusses the composition of two independent changes performed over a process model. Instance migration on the composite change is also discussed. Two classes of change composition are identified: 1) when the changes are made on disjoint regions of the model; 2) when both changes regions overlap. For disjoint regions, the composition results from implementing the changes in a sequence. When the regions overlap, an overlapping classification is introduced from equivalent changes to minor overlapping. Migration strategies are provided for all classes.

The *external monitoring* dimension specifies if environmental information is necessary to resolve the exception. The need to reference such external information has already been identified by Basil et al [2005]. In this thesis it is suggested that not only diagnosis but recovery as well may require referencing external information.

The item *tools to determine the best solution* identifies any additional tools necessary to implement the best recovery solution. This affords linking the framework with external tools supporting the decision processes.

In the motivating example, after the second plane hit the south tower, the national operations manager realized the need to involve everyone collecting external information necessary to identify how many planes were possibly hijacked and where they would be heading: they decided to use a white board to display such information. *External monitoring* was necessary and the *tools to calculate the best solution* involved a white board. The adopted *communication type* was synchronous and the *collaboration level* addressed several persons solving the problem in a collaborative mode.

Furthermore and most important, the plan to overcome the situation was not defined for every control centre. According to the available airports and number of planes they had to land, controllers implemented different local strategies. In particular, at the Memphis control centre, all controllers followed their planes until landing, instead of passing the planes between them. Operators favoured collaborative mode with respect to coordination mode to reduce response time.

Some empirical relationships between diagnosis and handling strategies where derived to support users deciding on the most adequate strategy that should be adopted for a particular situation. However, since these relations were not formally validated they are included in Appendix C and no further discussion is made on the subject.

## 4.7 Summary

In this chapter, we have established our conceptual approach, based on the extended version of the WFMC’s reference model, to support unstructured activities, relying on the organizational trajectories of the exception handling procedure and the organizational escalating concept.

The solution’s complete state diagram, regulating the five consistency levels identified in the previous chapter, was also defined. We have settled our focus on the consistency level 5 – supporting unstructured activities.

The basic functions that a level 5 system should support were then identified as: detection, diagnosis, recovery and monitoring. Moreover, diagnosis is not considered finished at the first approach and an intertwined play between the diagnosis on one side, and the recovery and monitor functions on the other, improve the situation understanding.

The taxonomy used to classify the event was also established. This taxonomy supports users on situation diagnosis and on the classification of the exceptional situation. The recovery techniques were also classified according to the adopted strategies users may choose during the exception handling process.

# Chapter 5

# Solution Architecture and Implementation

In this chapter we describe the solution’s architecture and its integration with the environment, the WfMS and the actors involved in the exception handling process. The architecture is derived from the extended reference model described in Section 4.1 and is integrates four components: *Exception Description*, *WF Interventions*, *Collaboration Support* and *Exception History*.

After establishing the architecture, the solution implementation by a dedicated workflow is discussed. The workflow implements the exception handling process described in the solution’s state diagram of Figure 4.1, including the basic problem solving functions: diagnosis, detection and monitoring. The remaining function, detection, is implemented by User Interfaces (UI) in the case of manual detection, or by specific workflow constructs inserted into the model and designed to automatically detect specific exceptional events. The implementation uses the OS open source suite of components. Implementation details are also described.

Therefore, in Section 5.1 we establish the solution architecture and the interfaces with the WfMS and the environment. Section 0 is dedicated to describe the exception handling workflow. Then, in Section 5.3 we describe the mechanisms used to implement automatic exception detection. Instantiating the exception handling workflow in manual and automatic exception detection is also described. Section 5.4 proceeds with the recovery and monitoring operations that users may implement during unstructured activities. The impact of the operations on the instance's consistency is also discussed. Section **Error! Reference source not found.** is dedicated to the implementation details. We start by introducing the OS suite and then explain how exception recovery and handling are implemented. Section 5.7 describes user interaction with the exception handling service by using UI examples. Finally, in Section 5.8 describes the exception data model where the exception related information is stored.

## 5.1 Architecture

To introduce the solution's architecture we will repeat in Figure 5.1 our extended reference model. However, the figure has been reorganized to place the Exception Handling Service at the top. Other components were readjusted in conformity. Additional detail was also added to the figure. In the architecture design, we have mainly focused on the interface with Workflow Enactment Service because our objective is to control the system behaviour at runtime. The monitoring functionality implemented by the Administration and Monitoring Tools component is important to collect data from the system. We assume the existence of primitives to implement this functionality if the solution is to be implemented on a generic WfMS. The administration facet of this component is not relevant since we did not study how to administer the exception handling system. We will also assume that the user has access to a Process Definition Tools component if, during the exception handling process, a new model should be issued. Primitives to transfer the model to the enactment services are also important and it is assumed that they exist, as usual on common WfMS. Finally, we have not studied the interoperability with other enactment services. The components under study are highlighted in the figure.

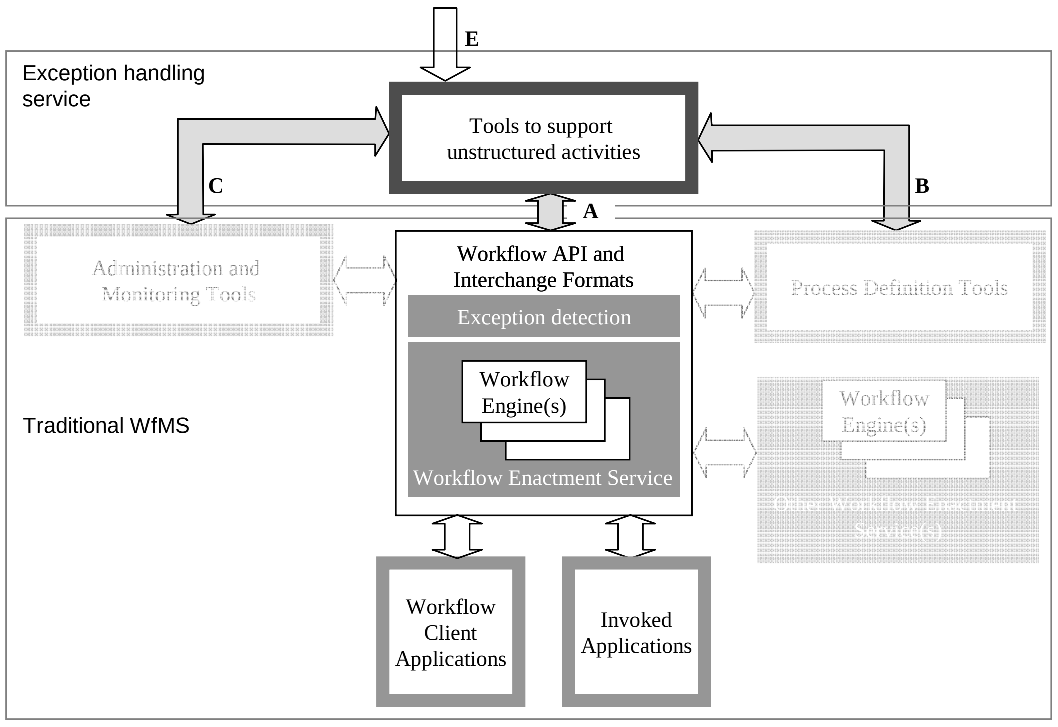

E
Exception handling service
Tools to support unstructured activities
C
B
A
Administration and Monitoring Tools
Workflow API and Interchange Formats
Exception detection
Process Definition Tools
Workflow Engine(s)
Workflow Enactment Service
Workflow Engine(s)
Other Workflow Enactment Service(s)
Traditional WfMS
Workflow Client Applications
Invoked Applications

Figure 5.1. Detailed version of the extended reference model reorganized. The external interface E and an exception detection component placed close to the enactment services were added.

Figure 5.2 is a detailed view of the highlighted components of Figure 5.1 comprising the Exception Handling Service, the Workflow Enactment Services represented by the workflow engine, the Exception Detection Component, and the workflow client and invoked applications represented by the tasks. Interface A and E are also illustrated. Dashed lines represent information flows whereas uninterrupted lines represent control flows.

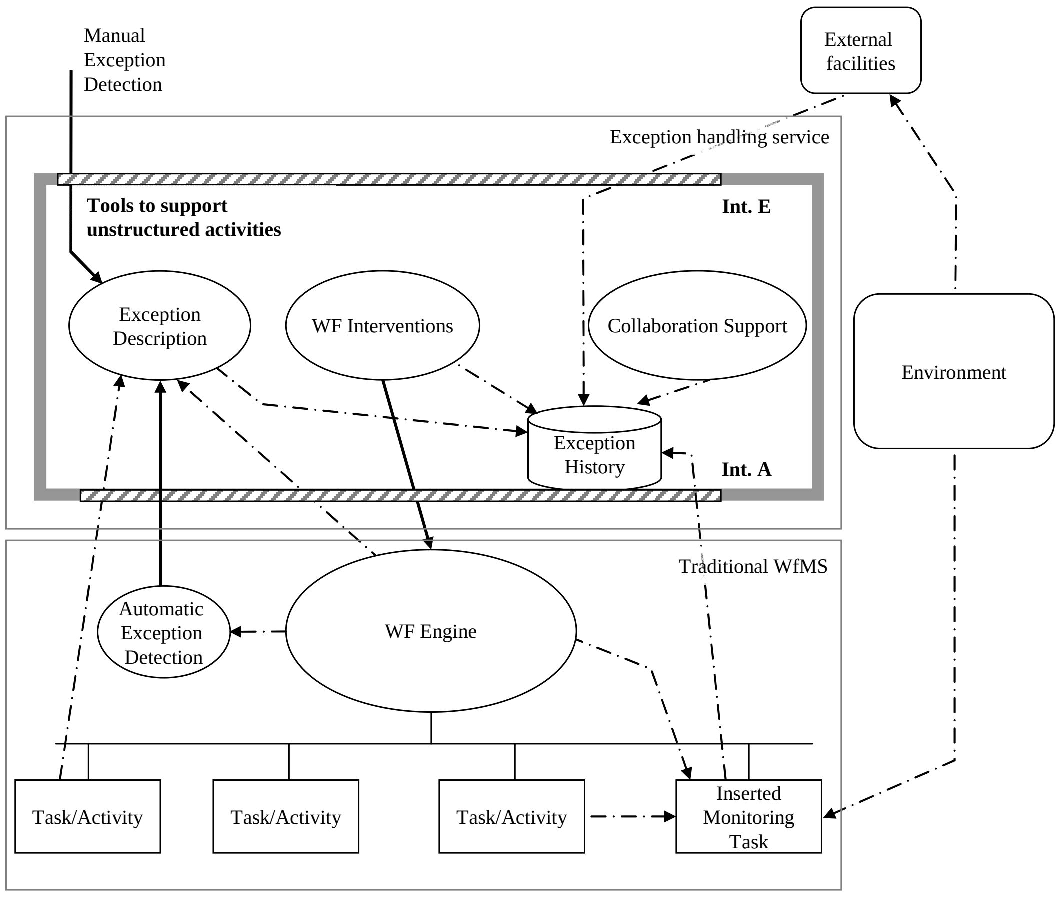


Figure 5.2. Solution’s architecture and its integration with the WfMS and the environment

Four components are identified as belonging to the exception handling service: *Exception Description*, *WF Interventions*, *Collaboration Support* and *Exception History*. Two distinct interfaces are also identified: interface A and E. The External Facilities, illustrated at the top, represent any exception handling activity carried outside the solution’s boundary and will be discussed bellow together with the interfaces.

The *Exception Description* component supports the diagnosis process described in Section 4.5. The *WF Interventions* component implements the functions associated to objective of the intervention described in Section 4.6. The *Collaboration Support* component implements the communication type and collaboration level mechanisms also described in Section 4.6. Finally, the *Exception History* component stores all relevant information associated to the exception handling cycles. Implementation details on these components will be presented in the following sections.

The traditional WfMS supported by the proposed solution is represented at the bottom of Figure 5.2. Naturally, the workflow engine plays a central role. Close to the engine, the *Automatic Exception Detection* component collects information from it and, when an exception is detected, control information is transferred to the Solution's Architecture. The *Inserted Monitoring Task* at the bottom right represents a task to collect information that users decided to insert during the exception handling activities. (It does not belong to any process being executed by the system.)

Concerning interfaces, the interface A links the exception handling components with the WfMS, while interface E links these components with the users and external environment. Interface A is used to collect information about the WfMS status, to implement low level recovery actions (launch/suspend tasks, etc), and to automatically detect and signal exceptions.

Interface E connects the exception handling service with users to enable manual exception detection and interaction during exception handling activities. Details on this interaction are illustrated bellow. Interface E also connects to External Facilities supporting environmental information gathering about the operations carried outside the framework's scope. Concerning environmental information gathering, remember that our discussion about completeness requires users not be restricted to the framework itself. Two types of activities carried out in the external context are differentiated: 1) information gathering, collaboration and decision making; and 2) recovery actions. The former are related with the use of external communication, coordination, collaboration and decision making tools (e.g., meetings, telephone conversations and operations research techniques). The later address any external recovery actions necessary to resolve the exception. It is our aim that, for any activity executed outside the framework's scope, some environmental information is inserted in the system for logging and monitoring purposes.

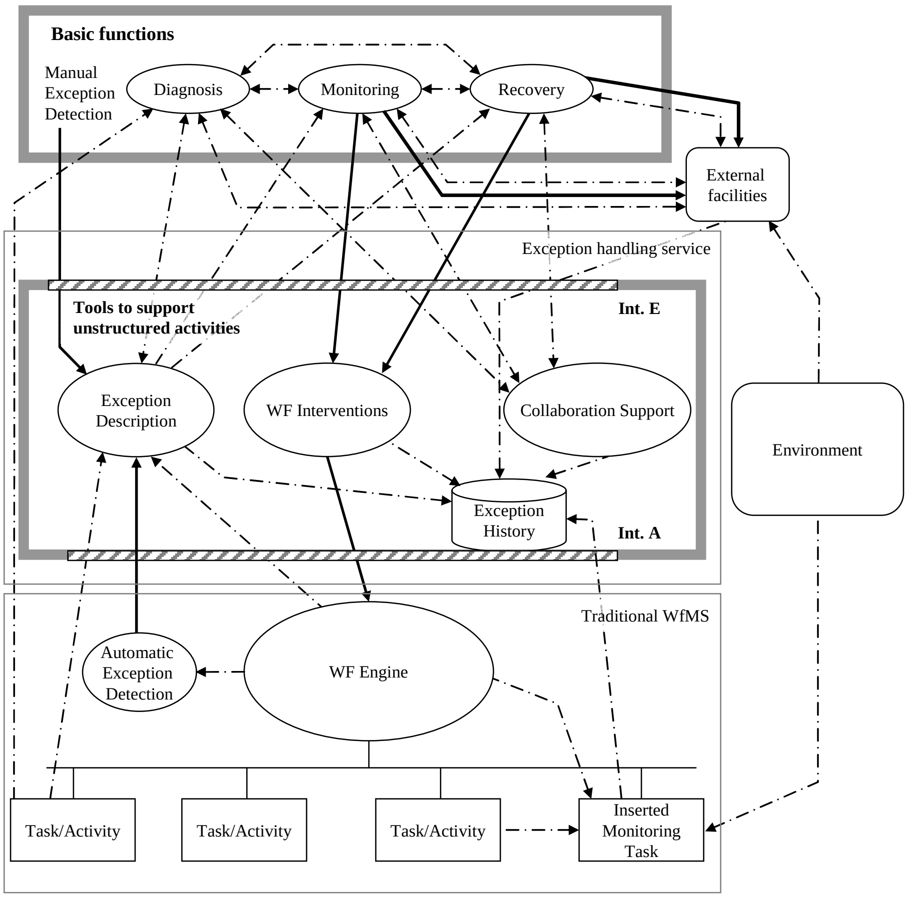


Figure 5.3. Solution's architecture and its integration with the WfMS, environment and operators' basic functions

In Figure 5.3, the three remaining operator’s basic functions (cf. Section 4.3) are added: diagnosis, recovery and monitoring. The diagnosis, recovery, and monitoring functions are carried out by the involved actors with support and orchestration from the components available at Tools to Support Unstructured Activities. Figure 5.3 represents the information and control flow through interface E between the operators and the system and between External Facilities and the system.

## 5.2 Exception handling workflow

Ellis and Keddara [2000] state that a process change is itself a process that can be modelled. Therefore, like Sadik [2000c], it is claimed in the present thesis that it is better to cope with ad hoc effective unexpected exceptions in work models using work models. The occurrence of an exception starts the exception handling workflow modelled in Figure 5.4.

This section discusses how this workflow implements the service components during the exception handling activities. Exception detection, the details of task implementation and UIs will be discussed throughout this chapter.

When an exceptional event is triggered, the system instantiates the exception handling workflow process and initialises some of the exception description parameters described in Section 4.5. There are two alternative ways to instantiate this process: either by system (interface A) or by user detection (interface E). They have been separated because these two tasks initialize the process in different ways. A responsible person must always be identified during detection (cf. Section 4.5). For system detected exceptions it is assumed that the responsible may not be the most indicated person to describe the situation. Therefore, the task Edit Info First Responsible is available right after detection where user defined as responsible by the system has the opportunity to define the new responsible and proceed to the following task: Edit Exception Info. For user exceptions this task is the first task after detection.

The purpose of the Edit Exception Info task is to specify some event parameters that the system or the user was not able to specify, or that should be redefined by the person responsible. For system detected exceptions, there is data that the system is not able to initialise and context information that requires human interpretation, e.g., the root cause falls in the first case, while the list of affected instances and person responsible fall in the second case. For user detected exceptions, the responsible person (that might not be the same person that triggered the exception) can redefine some of the parameters.

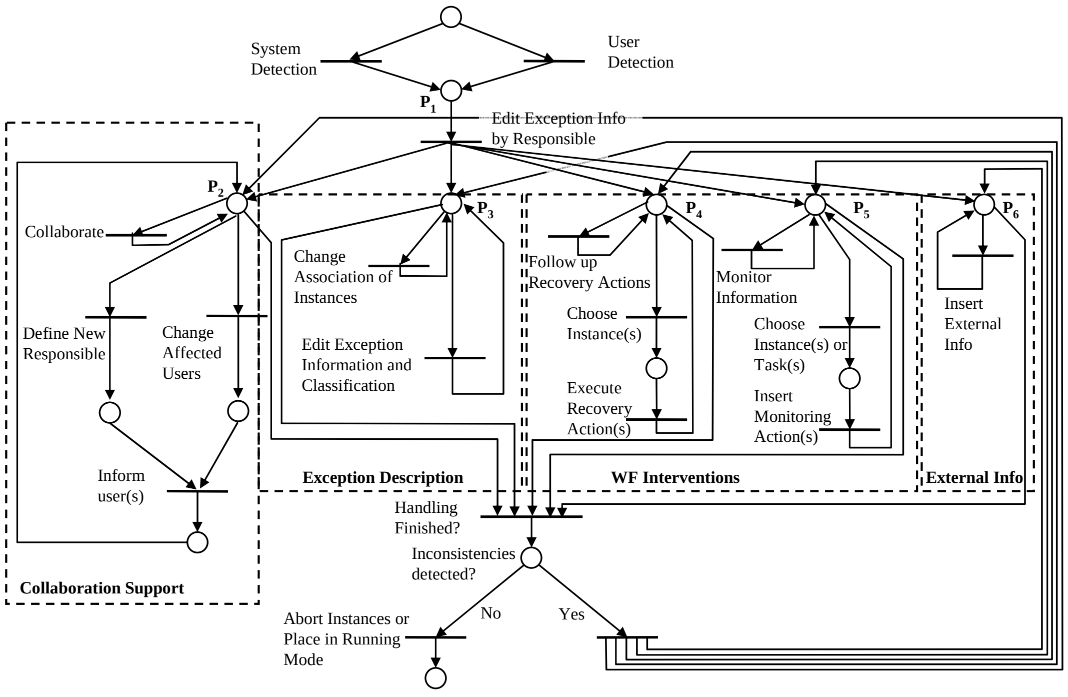

System Detection
User Detection
P1
Edit Exception Info by Responsible
P2
Collaborate
Define New Responsible
Change Affected Users
Inform user(s)
Collaboration Support
P3
Change Association of Instances
Edit Exception Information and Classification
Exception Description
P4
Follow up Recovery Actions
Choose Instance(s)
Execute Recovery Action(s)
P5
Monitor Information
Choose Instance(s) or Task(s)
Insert Monitoring Action(s)
WF Interventions
P6
Insert External Info
External Info
Handling Finished?
Inconsistencies detected?
No
Yes
Abort Instances or Place in Running Mode

Figure 5.4. Exception handling workflow

After this task the system enters in five parallel threads:

- Collaboration support;
- Exception description;
- WF Interventions;
- Insert external info.

The sub-models delimited in the figure by the dashed rectangles implement these components. The *WF Interventions* is implemented by independent recovery and monitoring threads. The interface E, identified in Figure 5.3, is implemented by the thread External Info.

The *Collaboration Support* component supports users specifically collaborating within the scope of an exceptional event. The tasks implemented by this component (see Figure 5.4) enable involving more actors in exception handling and implement the collaboration mechanism. The *Collaborate* task implemented by this component can be synchronous or asynchronous and at any time the users may choose which type to use. When asynchronous collaboration is used, any involved actor can send a message to any or all of the colleagues handling the exceptional event. The company email system or any other asynchronous alert system available in the organization may be used to notify the users that they should check the WfMS. Synchronous collaboration support depends on the application domain, since it can be implemented by a phone conversation, mobile phone messages, chat over a computer or even face-to-face conversations. In any case either the exchanged information or references to the collaborative actions are stored in the *Exception History* component. If it is not possible to automatically integrate this information, the users are requested to insert such references and any special additional comments. Further developments of this collaborative component including integration with existing collaboration technologies will be subject to future research.

The *WF Interventions component* is implemented with two threads: implement recovery actions and insert monitoring tasks. The specific actions implemented by this component enable users to implement recovery activities to bring the system back into a coherent state. They will be described in Section 5.4.

The monitoring thread affords users to insert monitoring tasks that store exceptional relevant information in the *Exception History*. Since this information is chronologically stored, the user may monitor the system evolution. Finally the *External Info* thread affords users to insert environment relevant information in the *Exception History* that can be lately used in the decision making process.

Exception handling activities proceed until the system is placed back in a coherent mode. When users identify that coherency has been achieved, they execute the task Handling Finished? (at the bottom of Figure 5.4) removing the marks from places $P_2$ to $P_6$ and suspending the support to unstructured activities.

Before finalizing the exception handling process, it is necessary to verify whether inconsistencies were inserted into the affected instances (cf. Section 4.2). If that is the case, the system must continue the support to unstructured activities and a mark is placed again on places $P_2$ to $P_6$, activating again the parallel threads. This is the last test shown in the model of Figure 5.4.

The discussion about the system usage can be enforced by using scenarios. In the remaining part of this section two examples are used. In the first case, the usage of the components are discussed regarding the involvement of some more actors in the exception handling procedure without any concrete scenario (organizational escalation), while in the second the 9/11 motivating example is used.

For the first example, if a user decides to involve more actors in the exceptional event s/he uses the *Collaboration Support* component. In the task Change Affected Users the new actors are associated to the exception. Then, in the task Inform Users, they are informed (e.g., email or mobile phone messages) that there is an exception to be collaboratively resolved. The diagnosis phase proceeds using the Collaborate task, so the other actors share their views of the present situation. Finally, they decide to insert two monitoring actions in the work model and two of them will be responsible for the follow up. Once any special event regarding these monitoring actions is triggered, the group is informed and the recovery action may proceed. The process is repeated until the exceptional situation is overcome and handling activities may finish. Figure 5.5.a) illustrates the organizational escalation for this exception handling process where the first affected user involved more actors from the same department in the handling effort. Figure 5.5.b) illustrates the organizational trajectory that initiates in the system where the process was being executed, and proceeds with the operator that signalled the exceptional event. Operators carry on the handling procedure until the event is overcome, when the control is placed backing the system.

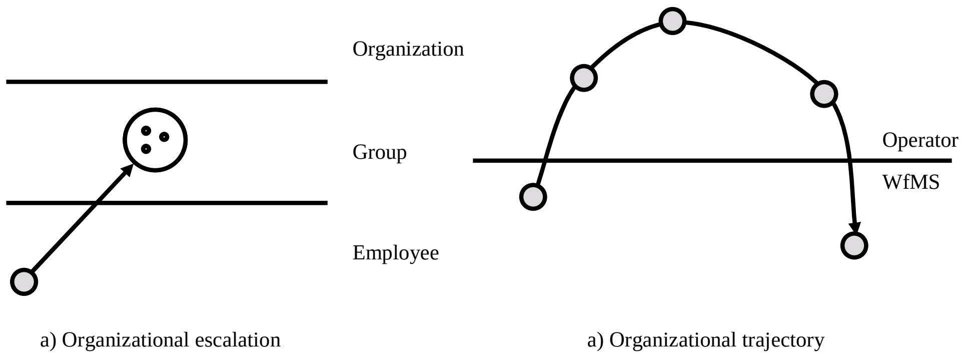


Figure 5.5. Organizational escalation and trajectory for the first scenarion

On the 9/11 motivating example, as soon as the first hijacked plane stops answering the controller calls and its transponder signals disappear from the screen a new exception is manually signalled. However, since it is an expected exception it is under model control with a special procedure. Only when the voice in the cockpit saying “We have some planes” is heard the exception is migrated to unexpected exception and the workflow is instantiated with the plane as the affected instance, and involving the controller, the supervisor and the national operations manager. The Collaboration Support component is used to escalate the exception to the supervisor and to the national operations manager, who becomes in charge of the exception handling.

When the second plane hit the south tower, the FAA’s command centre uses the Exception Description component to change the reaction time to quick, the difference to organization rules to a true exception and complexity to high. They use the collaboration component to involve all relevant people in the event.

The first nationwide recovery decision was to stop all takeoffs. This decision could have been spread through the WF Interventions component, affecting all upcoming instances, i.e., no new instance could be initiated. On the other hand, a new monitoring task was started on the entire organization: get information about any suspicious plane. Again, this could have been done within the architecture boundaries. A new task could have been started using the WF Interventions component, involving every control centre, to collect their information: they could only insert a single line of information for every suspicious plane. The information received from the control centres could be displayed on a large screen using a data show, relieving the personnel at the control centre in Herndon from collecting this information and writing it the wall. Even further, if a plane stopped from being suspicions, the associated line would disappear from the screen as soon as the associated control centre deleted it.

Some other local activities could have also been supported by this system after the action to land all planes was decided; e.g., by involving the neighbourhood airports on monitoring tasks, the air traffic controllers could decide on the fly what planes to route to each of them and airport personnel would be informed on line. On the other hand, airport personnel would inform the air traffic controllers on the available capacity to improve their decisions.

## 5.3 Automatic and manual exception detection

The Automatic Exception Detection component is placed close to the workflow engine in Figure 5.3 to collect information about executing tasks, workflow control data and workflow relevant data to detect exceptions. When an exception is detected, the Automatic Exception Detection component transfers the control through interface A to the exception handling system by instantiating the workflow discussed in the previous section. The workflow is instantiated by firing the System Detection task. As discussed in Section 4.4, the exception type's workflow, data, temporal and system/application can be automatically detected. The implementation of identification mechanisms for each of these types will be described in this section in terms of the modelling constructs used in the detection and the initialisation of the describing information. External and non-compliance exception types are manually detected and instantiate the workflow using the User Detection task in the model. The same process is used on the above mentioned types when detected by an operator.

The detection of workflow events is performed by pre-conditions and post-conditions structures (cf. Section 2.1.1). Section 2.3.1 identifies four types of workflow events: start/end of an instance; and start/end of a task. For the start of an instance a pre-condition is inserted on the first task of the model whereas for the end of an instance a post-condition is inserted at the last task. The task start situation is detected by a pre-condition in the task and the end task by a post-condition. E.g., if the modeller wants to assure that a loop is not executed more than *n* times a pre-condition can be inserted in the first task of the iteration. When the iteration counter reaches *n* the pre-condition triggers the Automatic Exception Detection component that instantiates the exception handling workflow. Information describing the event (cf. Section 4.5) is associated to the exception identifying the executing task, the affected instance, the user responsible, the reaction time and type of situation that triggered the event: e.g., the affected task, if it is a loop structure or if there is a deadlock. For manual tasks the user responsible is the person executing the task while automatic task have always an associated user that is the responsible person for any exception. The reaction time is defined in the post-function and is therefore fixed for the model.

Post-conditions are used to identify the presence of a data exception on the completion of a given task. Post-conditions detect the error on the data and trigger the automatic detection mechanism that instantiates the exception handling workflow. A special type of data exceptions affecting more than one instance was identified in Section 2.3.1. To detect these exceptions cross-instance checks must be carried out by the post-function. As with workflow events information describing the situation is associated to the exception where for data events the involved data structure is also kept. The process to select the responsible user and to set the reaction time is the same as in the previous case. Finally, the identification of all affected instances is also necessary when more than one instance is affected by the event.

In Section 2.3.1 three classes of temporal events were identified: timestamp, periodic and interval. The identification mechanism for each of these classes will be described separately.

The method used to identify the interval class, is depicted in Figure 5.6. Assume that the workflow designer would like to define a time interval constraint between tasks 1 and n of Figure 5.6.a. Figure 5.6.b shows how the workflow specification has been changed to incorporate that constraint. If $Task_n$ is executed before $T_1$ fires, the constraint was respected and no temporal event is triggered. However, if $T_1$ fires before $Task_n$, a token is placed on $P_2$ and the system triggers an exceptional event. The transition $T_2$ implements the same task as transition $Task_n$ and is inserted in the specification to assure that the workflow execution will not stop on $Task_n$ if an interval event is triggered.

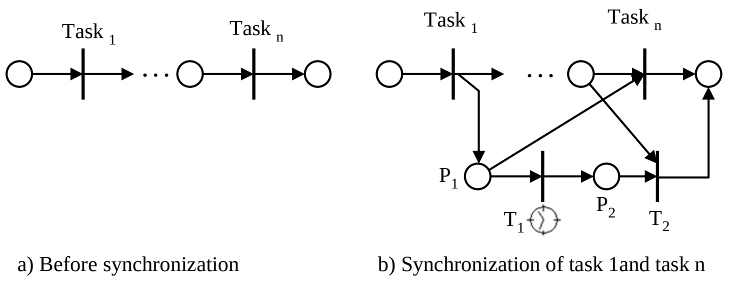

Task 1
Task n
Task 1
Task n
P1
P2
T1
T2
a) Before synchronization
b) Synchronization of task 1and task n

Figure 5.6. Identification mechanism for the interval class

When $T_1$ fires the exception handling workflow is instantiated. The affected instance can be suspended or allowed to continue depending on the specific handling activities. If the model is kept running, when the flow of work reaches the place before $Task_n$, $T_2$ will be enabled instead of $Task_n$. After $T_2$, is executed the work proceeds as normal.

For the timestamp the detection mechanism follows a similar scheme, where $Task_1$ is the initial task and $Task_n$ is the task identified in the timestamp. In this situation the timer is fired when the predefined date/time is reached. The exception handling workflow is instantiated as in the above example.

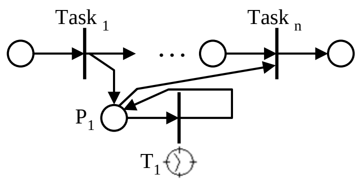


Figure 5.7. Identification mechanism for the periodical class

Figure 5.7 shows the Petri net used to implement the triggering mechanisms for the periodical class. The original model is shown on the top (tasks 1 and n, as in Figure 5.7.a) where task 1 is the first task of the workflow and task n is the last one. Place $P_1$ and transition $T_1$ where inserted to implement the periodical class. While the instance is running the timer is also running. When the time specified is reached one periodical event is triggered and the timer restarted again. The timer stops with the firing of the last transition in the workflow. Once again, when the transition $T_1$ fires the exception handling workflow is instantiated.

As in the previous events, information is associated to the exception identifying the timer, the instance and the responsible person.

Finally, the *system/application events* have characteristics similar to external events [Mourão and Antunes, 2003a], although, in some circumstances, the exception may be automatically identified, e.g., the system is able to identify that a database server stopped without requiring human intervention. For these events, the describing information is obtained from the underlying system where the event was generated while the responsible person is one user identified as the systems supervisor. For system exceptions all instances are affected and reaction time is set to quick. Instance execution is not suspended. For application exceptions it is possible that the affected instance can be detected. However, in some situations that is not the case. If the affected instance can be identified it is associated to the exception and time is set according to the global properties of the workflow model (i.e., global properties define if the instances are critical). If the affected instance can not be identified, all instances are affected to the exception and time is set to quick.

*External events* are a particular category of events, because they cannot be detected by the system as mentioned before. Thus, this type of event must be triggered by a human or by an external application. As mentioned in the beginning of this section, the instantiation of the exception handling workflow is achieved using the User Detection task in the model of Figure 5.4. For human detected exceptions, the operator is asked to initialise the exception related information through a dedicated UI, whereas for external application detection a public method is used where all the mandatory information is issued as parameters. This method uses the value in reaction time to select the way to inform the responsible person of the presence of a new exception.

Finally, *non-compliance events* correspond to situations where the desired process either deviates from the model (by requiring some special treatment) or the model is not applicable to a particular context. In this type of situation the system requires some additional information regarding the model, i.e. additional tasks, tasks that should be modified or removed from the model, etc. Due to the intrinsic nature of these events, they are dependent of the specific context that must be assessed by a human. Furthermore, these events may affect several tasks and processes. As in the previous scenario the user initialises the mandatory information using a dedicated UI implemented by the task User Detection in (cf. Figure 5.4).

## 5.4 Recovery actions, monitoring actions and support users removing inconsistencies

In Section 4.6 the high level objectives of the intervention were integrated into the handling strategies. To support users implementing these objectives, a set of quasi-atomic recovery actions are available. Involved actors may also implement recovery actions before deciding the most adequate objective for the entire case, e.g., to react to some partial erroneous condition even before the whole recovery procedure is decided. These quasi-atomic recovery actions increase operator's latitude during unstructured activities support. The Recovery Actions thread shown in Figure 5.4 affords operators to implement this functionality on the selected workflow instance(s). The responsible person first selects the workflow instance(s) and then chooses one of the available actions to apply over them. The user may only select instances that are in the same state because the state affects the consistency results of the operation. This way, the consistency results are valid for all selected instances.

The impact of implementing the recovery actions on instance consistency is also discussed. At the end of the section, we describe how to support users removing inconsistencies when the instance is to be placed again under model control.

Monitoring actions may also be implemented using the *ad hoc refinement* operation. Ad hoc refinement inserts threads executing in parallel to the actual instance execution and therefore constitute the natural choice to implement monitoring. Figure 5.8 illustrates the introduction of the monitoring task $T_{monitor}$ in parallel to the actual executing task $T_{n+1}$. The marked place $P_n$ indicates that the proceeding task is available for execution. Therefore, the insertion of a new thread with the monitoring task is equivalent to the transformation in the figure. Notice that task $T_m$ within the model is chosen as the join for both threads.

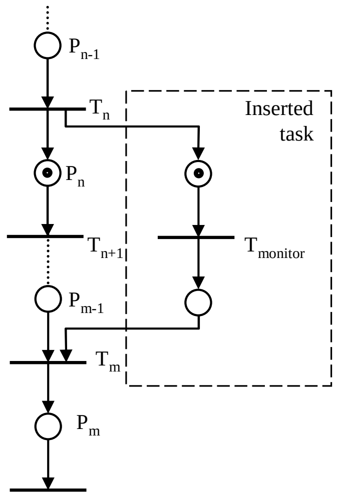


Figure 5.8. Inserting monitoring task

The following list of actions is currently available in the implemented solution [Eder and Liebhart, 1996; Agostini et al., 2003; Reichert et al., 2003; Chiu et al., 2001; Sadiq, 2000c]:

- *Suspend/resume instance* – this action involves suspending or resuming instance execution;
- *Abort instance* – abort the instance. As mentioned in Section 4.6 the instance can be aborted in one of two ways: hard or compensate some tasks;
- *Backward jump* – jump to a previous executed location in the work model. Some already executed tasks may have to be compensated;
- *Forward jump* – jump forward to a task in the work model;

- *Jump* – jump to another location in the model (this location is neither in the previous executed tasks nor in the upcoming tasks);
- *Move operation* – move one task to another location in the model;
- *Ad hoc refinement* – execute one action from a pre-defined list;
- *Ad hoc extension* – choose a new path or change the model.

Further implementation and functionality details about the recovery actions are the subject of the remaining part of this section. When needed, modelling constructs based on Petri Nets are used to discuss how the actions are implemented. The impact of the intervention on model consistency is also investigated. As mentioned in Section 3.4, metamodel assumptions based on Petri Nets will be used to derive conclusions. The same logic could also be used if some other metamodel assumptions were used even tough the conclusions might differ. The tool used to investigate the impact on model consistency does not impact our solution state diagram (cf. Figure 4.1 later refined in Figure 4.5 and Figure 4.6). Whatever the tool used, if the user is supported on detecting and repairing inconsistencies the state diagram is still applicable. Therefore, since we have already showed that such a tool exists, its implementation is not further discussed.

With the *suspend/resume* action the responsible person can suspend the execution of a specific instance. Later on, by issuing another action, the instance can be changed to the running state. During the suspended state no tasks can be initiated. However, the tasks that have already started are not aborted by the system. The persons attached to those tasks are informed of the situation. These operations do not affect the consistency of the running instances.

The action to *abort an instance* is to change a workflow instance to the end state to assure that no more actions defined in the workflow model will be executed. If some tasks require compensation activities they should be identified by the involved actors. The compensation task is instantiated in the exception handling workflow using the quasi-atomic action *ad hoc refinement* (described bellow). As in the suspend action above users attached to tasks that are being executed are informed on the situation. This operation does not affect the consistency of the running instances since they will be aborted.

*Backward jump* skips to a previous executed task, while *forward jump* skips forward to another task in the workflow instance. The required conditions to maintain model consistency will be highlighted and the discussion will proceed by analysing the impacts of not following the restrictions. As in [Reichert et al., 2003], *backward jumps* are jumps to actions in the history of the running instance.

Two illustrative backward jump situations are depicted in Figure 5.9. In $Jump_1$ the system reaches a deadlock on place $P_5$ because the set of places ($P_2$, $P_3$) does not have any marking. Therefore, place $P_3$ will never be marked and transition $T_4$ never fires. $Jump_2$ is correct because the target place of the jump is before the AND-Split implemented by transition T1. To avoid this situation the following criteria is defined:

> The subnet starting at the destination place ($P_1$ in Figure 5.9) of the jump and finishing at the original place ($P_6$ in Figure 5.9) can be isolated (including every node in every branch leading from the start place to the end and every arc that finish or start on those nodes.) From now on the delimited subnet will be referred as jump subnet;

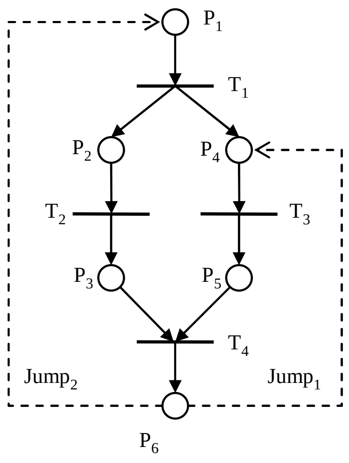


Figure 5.9. Backward jump

If the jump complies with the mentioned rules it is consistent because it does not insert inconsistencies in the model. (This rule is proofed in Appendix B.) On the other hand, if the jump is not compliant with this situation a reachability test must be performed, i.e., after changing the mark from its original place to the destination this new obtained marking condition is tested for reachability from the initial marking. If the marking is reached then the jump is consistent, otherwise it is not. It should be mentioned, as stated in Section 3.4., that verifying reachability is a complex problem that might not be solvable within a limited period of time. Therefore, if the model is not within any of the restrictions mentioned in the cited section, this computation might not be attainable. A time out should be used to prevent the user for waiting indefinitely. If the user decides to proceed with the jump even though it is inconsistent or there was a time out, this information is associated to the instance. This statement holds for every reachability test mentioned in this section.

Figure 5.10 represents the two different ways to implement a *Forward jump* as proposed by Reichert et al. [2003]: either the tasks in between are skipped (Figure

5.10.a) or executed in parallel with the tasks starting at the origin of the jump (Figure 5.10.b).

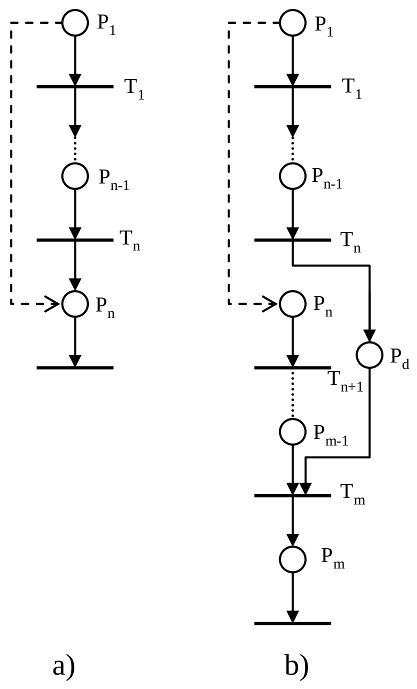


Figure 5.10. Forward jumps. a) abort tasks; b) parallel execution

If the tasks are to be skipped, the actual token is transferred to the destination of the jump. A check must be done to assure that the system does not run into a deadlock or a live lock. In this situation the jump is consistent if the subnet starting at the origin of the jump ($P_1$ in the Figure 5.10.a)) and finishing at the destination ($P_n$ in the Figure 5.10.a)) can be delimited (as defined above in the backward jump situation). It is assumed that the original net is consistent. (cf. Appendix B for the proof.)

On the jump in Figure 5.10.b) tasks are executed in parallel. In this situation an AND-Split is inserted on transition before place $P_1$ (not show in the figure) and a task $T_m$ must be selected to synchronize the two created parallel threads. The arc from $T_n$ to $P_n$ is removed and an AND-Join is inserted on task $T_m$ with arcs from $P_{m-1}$ and from a newly created place $P_d$. Note that this functionality requires modifying the model. Like in backward jumps, if the mentioned conditions for both forward jumps are not met a reachability test must be performed.

Figure 5.10 uses linear execution for simplicity; however the operation is consistent if the subnets from $P_1$ to $P_{n-1}$ and from $P_n$ to $P_{m-1}$ can be delimited (as defined above in the backward jump situation). (This rule is proofed in Appendix B.)

To implement forward execution of a task (as described in [Reichert et al., 2003]) the responsible person may use *ad hoc refinement* to execute the task and mark the task to be skipped (as mentioned bellow). This way, the task is executed as soon as necessary and skipped whenever reached during standard execution of the model.

The *jump* operation moves one mark to a location that is neither in the workflow history nor in the upcoming tasks. To verify the jump a reachability test must be performed on the target place, i.e., make a reachability test after moving the mark from its current location into the target. (The above mentioned characteristics of reachability tests should be taken into consideration when these tests are implemented.) If the state is reachable the jump is consistent, otherwise the user is advised on the test result to decide accordingly. If s/he decides to proceed with the jump an inconsistency is associated to the instance.

The *move operation* moves a block in the process to a new location, keeping the remainder of the model unchanged. A check to model consistency should be done after the move. Likewise the jump operation, after the test the user decides whether s/he wants to proceed. After the model is changed the system must also verify if the instance can be migrated to this new model and a reachability test is performed.

With the *ad hoc refinement* action the person responsible can choose to execute any activity from a list of standard ad hoc activities defined in the system. The list of existing ad hoc tasks currently contains a set of common WfMS tasks, such as making a phone call, sending an email and writing a letter. Users may even configure a task, based on a general task, that allows the edition of two database fields, inspecting the evolution of an Internet Web page (e.g., follow up the evolution of traffic conditions) and list the values from a database table.

Still considering *ad hoc refinement*, another list is made available with all the tasks defined in the affected processes. The user may then execute a task that was not yet executed or repeat the execution of a task already executed. To prevent unintended duplicate execution of a task executed in advance a marking mechanism is implemented that forces the task to be skipped when reached under model execution. The ad hoc refinement is not restricted. From [van der Aalst and Basten, 2002] a parallel thread can be initiated, executing other tasks and the final model consistency is not affected. Furthermore, this is a valid transfer rule with no deadlocks and proper completion.

This action is suitable to insert monitoring actions. When users identify that some data monitoring is required several times, they may insert the action in the ad hoc standard activity list.

*Ad hoc extensions* have a broader scope and a deeper impact on the workflow instance, since the person responsible can select an alternative path for the instances or perform small changes to the model. On the alternative path scenario two situations must be considered: 1) the new model will replace all the threads being executed on the old model and new model execution starts from the beginning with only one active thread; or 2) the transfer is more complicated because multiple threads are to be transferred to different locations in the new model. On the former case, if the new path is consistent the transfer will be consistent as well. On the later case, the user must identify the places to mark in the new model. A reachability check to new model will have to be performed to identify if the intended state is reachable. It is assumed that the new model is consistent.

When small interventions to the model are performed, the user should be advised to follow the restrictions identified in Section 3.4.2. If the user chooses to apply changes not predicted, consistency and reachability checks must be performed.

It should be noticed that the *move* and *ad hoc refinement* are particular cases of *ad hoc extensions* since they both result in executing a new version of the existing model. However, they are treated separately because they implement recovery mechanisms that have a different interpretation from the user perspective.

When the handling operation is finished, the system checks for any inconsistencies associated to the running instances if the user wants to place them again under model control. Only when all inconsistencies are resolved the system can be placed on running mode. The system first implements a consistency test to the resulting model. Model inconsistencies must be removed before any further action. Then, the system performs a reachability test on the affected instances. If the marked places are not reachable, the instance is inconsistent. The system presents the reachable sets that include at least one marked place in the actual state of the instance. The users may then implement any of the available recovery actions to remove the inconsistencies.

## 5.5 Applications Programmer Interface with the Service

In this section we define the list of functions that compose the Applications Programmer Interface (API) for the exception handling service. These functions may be used by other applications to access the service.

Functions were grouped according to the functionality they implement in: 1) escalation; 2) collaboration; 3) exception description; 4) WF Interventions; and 5) External info.

Functions in the escalation group:

- `AddAffectedUsers(List users)`

  Affect all users in the list `users` and return the resulting list of affected users.

- `RemoveAffectedUsers(List users)`

  Remove all users in the list `users` from the list of affected users and return the resulting list of affected users.

- `ChangeUserResponsible(String newUser)`

  Change the responsible to `newUser`. Return the new responsible if the change is successful.

Functions in the collaboration group:

- `SendEmail(List users, String subj, String content)`

  Send an email with the specified `subj` and `content` to the users whose username is specified in the list `users`. Return a list of all users the email was sent to. A link to the Web page that implements the UI for the current exception is added to the email content.

- `SendMobileMessage(List users, String content)`

  Verify the mobile number for all users in the list `users` and sends the a mobile message with the `content`. Return a list of all users the message was sent to.

- `SendInstanteMessaging(List users, String content)`

  Initiate an instant message interaction between the users within the list `users` by issuing the message in `content`. Return the list of users that are online in the instant message service.

Functions in the exception description group:

- `ChangeExceptionInfo(ExceptionInfoType newData)`

  Update exception related information.

- `AddAffectedInstances(List newInstances)`

  Add instances in the list `instances` and return the resulting list of affected instances.

- `RemoveAffectedInstances(List remInstances)`

  Remove all instances in the list `instances` from the list of affected instances and returns the resulting list of affected instances. The instances for which any recovery action has been implemented can not be removed.

Functions in the WF interventions group:

- `SuspendInstance(List instances)`

  Suspend all instances[11] in the list `instances` if they are affected by the exception. Return a list with all instances that were suspended.

- `Resume(List instances)`

  Resume all instances in the list `instances` if they have been suspended. Return a list with all instances that were resumed.

- `AbortInstances(List instances)`

  Abort all instances in the list `instances` if they are affected by the exception. Return a list with all the aborted instances.

- `BwdJumpInv(List instances, StateType newState)`

  Verify if the state `newState` is a state in the history of all instances in the list. Every instance must be based on the same model and be at the same state.

[11] In all functions of this group, the instances must be affected to the exception.

- `BwdJumpCons(List instances, StateType newState)`

  Verify if the backward jump to state `newState` is consistent for the instances in the list `instances`. Returns true if the jump is consistent and no otherwise. Every instance must be based on the same model and be at the same state.

- `BackwardJump(List instances, StateType newState)`

  Jump backward to the state `newState` that must be in the history of every instance. Every instance must be based on the same model and be at the same state.

- `FwdJumpInv(List instances, StateType newState)`

  Verify if the state `newState` may be a future state for all instances in the list. Every instance must be based on the same model and be at the same state.

- `FwdJumpCons(List instances, StateType newState)`

  Verify if the forward jump to state `newState` is consistent for the instances in the list `instances`. Returns true if the jump is consistent and no otherwise. Every instance must be based on the same model and be at the same state.

- `ForwardJump(List instances, StateType newState, Boolean paralExe)`

  Jump forward to the state `newState` that must be reachable from the current instances state. Variable `paralExe` indicates if the tasks in the middle are executed in parallel (`true`) or skipped (`false`). Every instance must be based on the same model and be at the same state.

- `JumpCons(List instances, StateType newState)`

  Verify if the jump to state `newState` is consistent for the instances in the list `instances`. Returns true if the jump is consistent and no otherwise. Every instance must be based on the same model and be at the same state.

- `Jump(List instances, StateType newState)`

Jump to the state `newState` in the workflow model. Every instance must be based on the same model and be at the same state.

- `ChangeInstanceModel(List instances, String newModel, list newStates)`

Jump to the state `newStates` in the workflow model. Every instance must be based on the same model and be at the same state. There may be more than one active thread in the new model may be more than one.

- `VerifyReach(List instances, String model, StateType newState)`

Verify if the state `newState` is a reachable state from the current state of the instances in the list. Every instance must be based on the same model and be at the same state.

- `VerifyReachFromStart(String model, StateType newState)`

Verify if the state `newState` is a reachable state in `model`.

- `StartParThread(List instances, String model)`

  Start a new parallel thread with `model` for every instance in the list.

- `ListAdhocTasks()`

  List the ad hoc predefined tasks.

- `ListInstanceTasks(List instances)`

  List the tasks defined in the model for these instances. Every instance must be based on the same model and be at the same state.

- `ExecuteParTask(List instances, Task task)`

  Start a parallel thread to execute `task`. Every instance must be based on the same model.

- `ExecuteParTaskInModel(List instances, Task task, Boolean repeat)`

  Start a parallel thread to execute `task` that belongs to the list of tasks available in the workflow model. If the task is later reached during standard execution, it is skipped if the parameter `repeat` is `false` and executed again otherwise. Every instance must be based on the same model.

Functions in the External info group:

- `InsertExternalInfo(String info)`

  Insert the string `info` into the external related information.

## 5.6 Implementation in the OpenSymphony platform

In this section we describe the solution implementation. In the following, we start with some relevant details about the system platform. Then, we identify the exception handling components integrated with the system platform. To finish this section, we present the implemented data model and illustrate the system use.

### 5.6.1. The OSWorkflow project

The OSWorkflow (OSWF) is a project within the OS [The OpenSymphony project, 2005] open source suite of components that implements a workflow engine. Other projects in the suite implement user validation to passwords and roles, a timer component, persistence store of workflow application data and Web interfaces. All the components are developed in Java and run over a servlet container. Workflow models are stored in Extended Markup Language (XML) files.

The OSWF project stores the workflow control data in a Relational Database Management System (RDBMS). Figure 5.11 represents the complete set of tables and their relationships in the referential model.

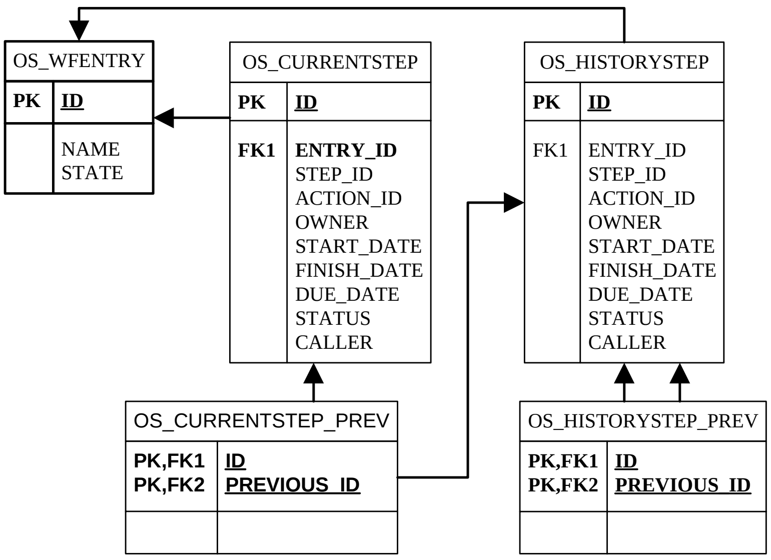


Figure 5.11. OSWorkflow referential model

An example is used to illustrate the execution and control of workflow instances by the engine. The table fields will also be described within the example. It is assumed that the file named “example.xml” contains the workflow model. Note that in the OSWF nomenclature workflow states as steps. As it will be explained bellow, there is not a direct correspondence in OSWF nomenclature to the concept of workflow tasks. Remember that in our adopted Petri Net modelling formalism, a workflow task includes task execution and state transition.

The main table, OS_WFEntry is shown in Figure 5.12 after the workflow instance has been initialized. The ID field is the key for the workflow instance, the NAME is the file with the model and STATE indicates whether this instance is activated, suspended, completed or killed.

| ID | NAME | STATE |
|---|---|---|
| … | | |
| 32 | example.xml | Activated |
| … | | |

Figure 5.12. OS_WFEntry table after the example initialization

When a new instance is created, the engine inserts a new row in the table with a generated ID field and the file name chosen by the caller. After successfully execution of the initialization routine, the field STATE is set to *Activated*. An example with the sequence of methods to create a workflow instance is shown in Code listing 5.1.

```
Workflow wf = new BasicWorkflow(username);
long id = wf.initialize("example", initAction, mapInputs);
```

Code listing 5.1. Create a workflow instance

The first method initializes the object and sets an internal variable with the name of the user logged on the system. The second method creates and initializes the new workflow instance. The first parameter is the name of the XML file with the model, the `initAction` variable indicates the number of the action to be executed and `mapInputs` is a set of key to value pairs used by the action. These methods belong to the OSWF API that defines the interface used by programs to access OSWF functions.

The XML model file can now be described to explain how the actions are executed by the workflow engine. The model element in Figure 5.13 represents the initial state for a generic workflow and is named “initial actions”. The figure shows the available initialisation actions numbered from 1 to n. Action 1 is presented in detail to allow the description of one action execution presented bellow. The action structure is the same for every workflow step and therefore this description is applicable.

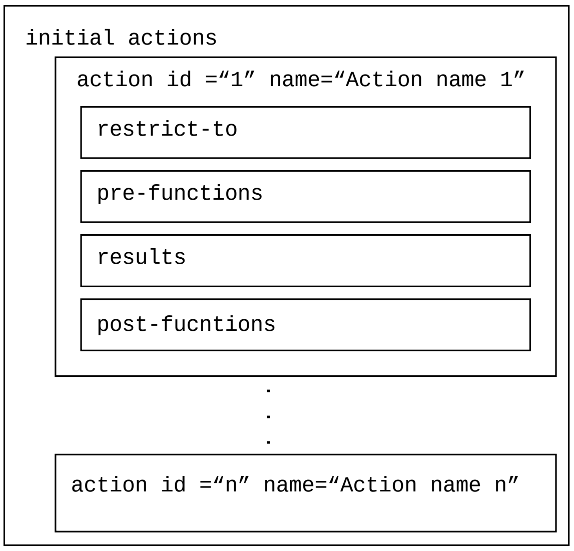

initial actions
action id ="1" name="Action name 1"
restrict-to
pre-functions
results
post-fucntions
action id ="n" name="Action name n"

Figure 5.13. Hierarchical organization of initial actions in a OSWF model

As mentioned before, a workflow state is named a step in OSWF. Figure 5.14 shows the hierarchical organization of the steps. Every step has an initial collection of *external permissions* that enables the definition of a set of conditions to control task execution. The conditions may also depend on workflow relevant data and workflow control data. Since the model allows multiple permissions more than one task can be active at the same moment. This construction may be used to control who has access to the task or what is the task to execute in the step according to workflow control data and workflow relevant data. It is also important to note that task execution can proceed as soon as the workflow is at the step, if any of the external-permissions elements evaluate to true. Even further, for as long as the workflow instance remains at the step, users can still execute the task. When the task finishes, an action must be executed to trigger step transition. This distinction between workflow task in the Petri Net modelling formalism and action in OSWF nomenclature should be emphasized. The execution of one task in OSWF does not trigger workflow step evolution. An action must always be executed when the task finishes changing the actual step number for the instance.

Therefore, the workflow Petri Net task (the transition) is said to be spread through the step and the action.

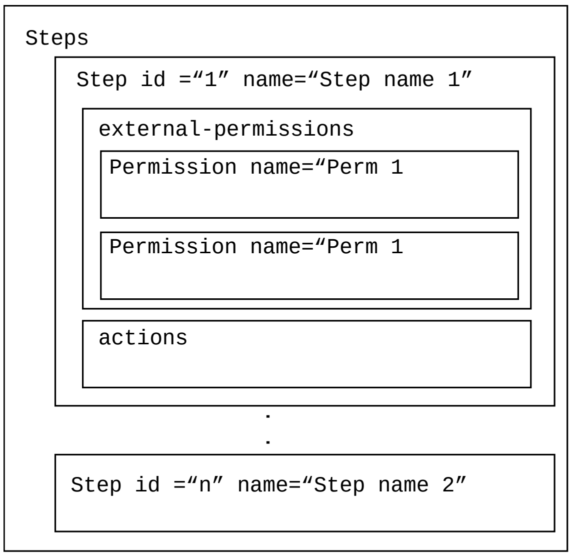


Figure 5.14. Hierarchical organization of steps in a OSWF model

Coming back to the initialization sequence, to execute action ID 1 (to avoid confusion this action will be referred as action 1) in the instantiation process, the variable `initAction` must be equal to 1 in the `wf.initialize` method of Code listing 5.1. The details for action ID 1 are also represented in the example model of Figure 5.13. Each action in an OSWF model can contain four distinct elements: *restrict-to*, *pre-functions*, *results* and *post-functions*.

The *restrict-to* element is composed by a series of conditions that must be evaluated to true to allow the execution of the action, e.g., only users that belong to a given role can execute the action. The following element is the *pre-functions*. These functions execute code before the state transition takes place. They are typically used to evaluate conditions to be used on OR-Splits or to return workflow relevant data that is to be stored in the OSWF tables.

The next element is named *results* and is used to control the transition, i.e., the next step for the workflow instance. Each element results can have zero or more *conditional results* elements but must have at least one *unconditional result* element [The OpenSymphony project, 2005]. This structure can be compared to a "case" statement in a typical programming language where the element "case else" is mandatory. The first conditional element that is evaluated to true is executed. If none of the conditional elements is true the unconditional element is executed.

An example is used to clarify how the attributes in the result element are used by the transition. Assume that there are no conditional results and the unconditional result in action 1 is the Code listing 5.2. The unconditional result indicates to the engine the number of the next step in the attribute `step`. The remaining attributes will be explained bellow during the description of the table that stores workflow control data. Therefore, after the element is executed by the engine the workflow instance is in step ID 1 (step 1 from now on).

```
<unconditional-result old-status="Finished" status="Run"
       step="1" owner="$(caller)"/>
```

Code listing 5.2. Unconditional result for action ID 1

Conditional result elements are similar to unconditional in the sense that they have the same information for step transition. They only have one condition at the beginning that is evaluated to verify if they are executed. It has been explained above that the *results* element included in the step can have several *conditional results* but must have at least one *unconditional result*. Therefore transition in the step state is always assured if an action is executed. On the other hand, this form of *conditional* and *unconditional results* correspond to an OR-Split, i.e., various conditional results being tested and only one defining the next step means that the direction of the flow is chosen by the executed element. The AND-Split has a slightly more difficult definition and is explained at the end of this section. At this moment, it is important to realise that when an AND-Split is executed the table OS_CURRENTSTEP will have two entries for the instance (one row for each parallel thread).

Finally, the last element in the action is the *post-functions* that are executed after the transition takes place (e.g., send an email to a user indicating that the action in the new step of the workflow instance is available to be performed.) Figure 5.15 is the execution flow of the elements defined in the model for one action and summarizes the above description.

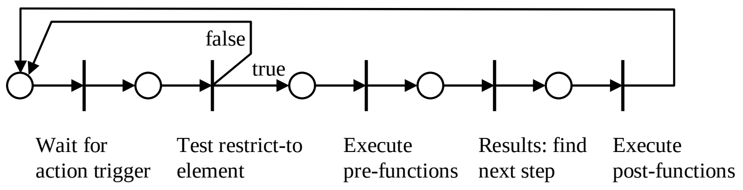


Figure 5.15. State transition of the OSWF engine

The information stored by the engine in the database will now be explained. The current steps of the various workflow instances running on the system (workflow control data) are stored by the OS_CURRENTSTEP table. Figure 5.16 lists the table field values after the successful initialization of the example using action 1. The ID field is the key for this table and is automatically generated. The ENTRY_ID field is the foreign key to reference the workflow entry table (the OS_WFEntry described above). The fields STEP_ID, OWNER and STATUS reflect the attributes specified in the *unconditional result* element. As mentioned before, the state assumed by the workflow instance after the transition takes place is specified by the attribute `step` (1 in Code listing 5.2) and stored in the field STEP_ID. The OWNER field is specified in the attribute `owner` and, assuming the username that triggered the initialization process was “João”, is the value in Figure 5.16. Finally, the attribute `status` specifies the field STATUS of the table and can assume any value. Fields OWNER and STATUS may be used in the *external permissions* element of the step (explained above) to control who can execute the defined tasks. Remember that more than one task may be defined and these fields may be used in conditions to specify what actions are available to what users.

| ID | ENTRY _ID | STEP _ID | ACTION _ID | OWNER | START_ DATE | FINISH _DATE | DUE _DATE | STATUS | CALLER |
|---|---|---|---|---|---|---|---|---|---|
| ……. | | | | | | | | | |
| 5 | 32 | 1 | | João | 4/4/2004 11:50:33 | | | Run | |
| …….. | | | | | | | | | |

Figure 5.16. OS_CURRENTSTEP table after the example initialization

The fields ACTION_ID, FINISH_DATE and CALLER are set to null because they will be used when the next action, executed on step 1, is executed. The DUE_DATE field can be used to set the desired due date for this task. To set a value for DUE_DATE the optional attribute `due-date` must be used in the unconditional result element in Code listing 5.2.

After the transition takes place and the post-functions are executed the instance becomes idle until another action is performed over it. In the example, the workflow is on step 1 waiting for any user-triggered action or any automatic action (the OSWF project has a special type of actions, called automatic actions, which are automatically fired when the engine reaches the step where they are defined).

Assume now, that action ID 3 (defined in step 1 of example.xml) is later executed by username “João” on 6/4/2004 15:30:45. The row in Figure 5.16 is copied to the OS_HISTORYSTEP table and a new row is inserted in OS_CURRENTSTEP table reflecting the results of action 3. Figure 5.17 lists the table OS_HISTORYSTEP. The figure shows the fields ACTION_ID, FINISH_DATE and CALLER with the values already settled and defined by the execution of action 3.

| ID | ENTRY _ID | STEP _ID | ACTION _ID | OWNER | START_ DATE | FINISH _DATE | DUE _DATE | STATUS | CALLER |
|---|---|---|---|---|---|---|---|---|---|
| ……. | | | | | | | | | |
| 5 | 32 | 1 | 3 | João | 4/4/2004 11:50:33 | 6/4/2004 15:30:45 | | Run | Joao |
| …….. | | | | | | | | | |

Figure 5.17. OS_HISTORYSTEP table after execution of action 2

To carry on the execution of the workflow, the engine has methods that return the actions that the logged user can perform on the workflow. These methods use the workflow ID to retrieve the model filename from the OS_WFEntry table and the actual step ID from OS_CURRENTSTEP table. Then, from the defined actions are retrieved from the XML model.

The last OS_CURRENTSTEP_PREV and OS_HISTORYSTEP_PREV tables identify the action executed before the current step and link the history of the tasks executed in the workflow respectively.

From the above description it is important to highlight there is equivalence between the Petri Nets modelling and the OSWF. The marked Petri Net places correspond to steps in the OSWF. However, in Petri Nets task execution is modelled by the transition whereas in OSWF task execution proceeds while the instance is at the step. Actions in OSWF are used to control the state transition between steps and not task execution. Therefore the Petri Net transition concept is spread through the step and the action. This fact introduces some minor adjustments when the developed models are written using the OSWF meta-language. Nevertheless, the equivalence between places and steps simplifies the overall implementation of the models.

Finally, as mentioned before, the AND-Split has a particular structure in the OSWF project and should be described. Code listing 5.3 is an example of an AND-Split defined for one example action. The unconditional result element in the action refers to the element split ID=1. In the split element, each unconditional result row element refers to one parallel thread that is to be generated and the attributes specify the associated information. In the example, two parallel threads are generated for steps ID=30 and ID=40.

```
<steps>
 . . . .
 <step id=”10” name=”Step Example”>
   <action id=”10” name=”Action example”>
     . . .
     <results>
       <unconditional-result old-status="Finished" split="1"/>
     </results>
   </action>
 </step>
 . . . . .
</steps>
<splits>
 <split id="1">
   <unconditional-result old-status="Finished" status="Underway"
         owner="$(caller)" step="30"/>
   <unconditional-result old-status="Finished" status="Finished"
         owner="$(responsible)" step="40"/>
 </split>
</splits>
```

Code listing 5.3. Example of an AND-Split in the OSWF project

The AND-Join has an equivalent structure since multiple threads where multiple steps are active must join into one step. It is equivalent in the sense that every step that is to be joined must have an action that refers to a Join element defined in a corresponding element. Code listing 5.4 is an example joining steps ID=35 and ID =45. To avoid duplication only step ID=35 is shown. The join element at the end shows a condition where each parallel thread to be joined is tested to their status.

If one of them is not in the required step ID or the status attribute is not set to finish the join does not execute. On the contrary, if both the threads in the example are in step ID=35 and ID=45 and their status is finished the join is executed merging all the threads into a single one with the step ID=60 active (unconditional result element at the end of the join).

```
<steps>
 . . . .
 <step id=”20” name=”Step Example2”>
  <action id=”100” name=”Action example2”>
   . . .
   <results>
    <unconditional-result old-status="Finished" join="1"/>
   </results>
  </action>
 </step>
 . . . .
</steps>
<joins>
 <join id="1">
  <conditions type="AND">
   <condition type="beanshell">
     <arg name="script"><![CDATA[
        "Finished".equals(jn.getStep(35).getStatus()) &&
        "Finished".equals(jn.getStep(45).getStatus())
    ]]></arg>
   </condition>
  </conditions>
  <unconditional-result old-status="Finished" status="Underway
          " step="60"/>
 </join>
</joins>
```

Code listing 5.4. Example of an AND-Join in the OSWF project

## 5.6.2. Exception detection and signalling in the OSWF project

In this section we describe the implementation in the OSWF project of the mechanisms listed in Section 5.3 to detect exceptions. Once the exception is detected, the exception handling workflow (cf. Section 0) is instantiated. The instantiation process uses the OSWF API as described in the previous section. The mechanisms to automatically detect the workflow, data, temporal and system/application exceptions are described. Exception related information is stored on a database described in Section 5.8.

As discussed in Section 5.3, pre-conditions and post-conditions are used to detect workflow events. Pre-conditions are implemented using the external-permissions element of the step. The inserted condition may also be used to instantiate the exception handling workflow. Post-conditions are mapped to conditional results elements. When the detection does not restrict workflow evolution, it is enough to insert a pre-function or a post-function to instantiate the exception handling workflow. Since pre-functions run before state transition they will be preferred. From now on, and to avoid confusion, the adopted detection mechanism is based on pre-functions. In Section 5.3 four workflow event types were identified: start/end of an instance; and start/end of a task. The start of an instance is detected by a pre-condition on the first task of the model. Therefore, in OSWF the initial actions element (refer Section 5.6.1) is used to detect the event and the workflow initialisation actions are transferred to a newly inserted step. The end of an instance uses a pre-function in the last task of the model. For the task start a new step must be inserted before, while for the task end situation a pre-function is inserted on the monitored task.

Exception related information (cf. Section 5.3) is initialised by the function used to detect the event. Since it is a dedicated function designed to detect the particular event, it embeds the appropriate describing information and the reaction time value. The remaining information is obtained by the function within the workflow engine, namely the affected instance, the step number and the responsible user. In all mentioned situations, after the exception handling workflow is instantiated, the affected instance state can be changed to idle on the same function that detected the event. Again, this information is embed in the function and may also depend on application data, workflow relevant data or workflow control data. Even further, since some tasks do not depend on the engine, the owner can be informed by a communication mechanism (either email or mobile message depending on the event). This functionality refers to all detection mechanisms described bellow.

*Data events* are implemented using the pre-function element of the action. Every data structure is stored in a database and managed in local memory using a dedicated class. Classes that support data structures implement a special method named validateData that evaluates data consistency according to a predefined set of rules. The tasks that manipulate data structures must evoke a special pre-function indicating the object to be checked as a parameter. The pre-function runs the objects method and instantiates the exception handling workflow on the presence of any data inconsistency. The object returned by the validateData method describes the reason for the exception and indicates if the instance is to be suspended. The description and the step number are stored in the exception describing information and the user that executed the task is defined as responsible. If the violated constraint results from a cross-instance check (cf. Section 5.3) all affected instance(s) that violate(s) the constraint are also identified in the object returned by the validateData method. This affected instance(s) are also suspended if indicated by the object.

All *temporal events* are supported by the Quartz project provided by the OS suite that implements a time triggering mechanism. The triggering mechanisms for the temporal events must be inserted in the workflow model where the exception is to be detected. Code listing 5.5 is an example for the periodical class detection represented in Figure 5.7. The easy mappings between places in Petri Nets and steps for one side and between tasks and actions for the other are illustrated by the example. In the initial actions element the timer is started and a split generates two parallel threads: one for the main workflow and another for the identification mechanism. Step ID=1000 models the place $P_1$ where two actions are implemented. The first action refers to the timer time-out and is invoked by the timer while the second is an AND-Join where the two parallel threads of Figure 5.7 are joined to terminate the workflow. The next step after the join is used to terminate the instance. The pre-function executed inside the timer time-out action instantiates the exception handling workflow. The other periodical classes have similar implementation.

```
<initial-actions>
 <action id="1" name="Action to initialize business process">
   . . . .
   <pre-functions>
     <function type="class">
        <arg name="class.name">com.hrm.util.InitialiseTimer</arg>
     </function>
   </pre-functions>
   <results>
     <unconditional-result old-status="Finished" split="1"/>
   </results>
 </action>
</initial-actions>
<steps>
 . . . .
 <step id="1000" name="Place P1">
   <action id="10000" name="Timer time out">
     <pre-functions>
       <function type="class">
            <arg name="class.name">
                 com.hrm.util.InstantiateException
            </arg>
       </function>
     </pre-functions>
     <results>
       <unconditional-result old-status="Finished"
                 status="Finished" step="1000" owner="$(caller)"/>
     </results>
   </action>
   <action id="10010" name="WF finished">
     <results>
       <unconditional-result old-status="Finished" join="1"/>
     </results>
   </action>
  </step>
  . . . .
 </steps>
 <splits>
  <split id="1">
    <unconditional-result old-status="Finished" status="Underway"
          owner="$(caller)" step="Task1"/>
    <unconditional-result old-status="Finished" status="Finished"
          owner="$(responsible)" step="1000"/>
  </split>
 </splits>
 <joins>
  <join id="1">
    <conditions type="AND">
      <condition type="beanshell">
         <arg name="script"><![CDATA[
            "Finished".equals(jn.getStep(Taskn).getStatus()) &&
            "Finished".equals(jn.getStep(10000).getStatus())
       ]]></arg>
      </condition>
    </conditions>
    <unconditional-result old-status="Finished" status="Finished
               "step="Terminate"/>
  </join>
 </joins>
```

Code listing 5.5. Detection mechanism for the periodical temporal event

Exception information is initialised in the InstantiateException function that is executed when the timer fires and step named Place $P_1$ is active. Like in the previous scenarios, the information identified in Section 5.3 is associated to the new exception.

*System and application* events are identified using the catch mechanism of the Java programming languages. If during code execution a non-caught exception construct from the program is raised, the code instantiates the exception recovery workflow. If the error is originated on the underlying systems supporting the workflow engine a system exception is raised, and if it is originated on the applications that implement the tasks an application exception is generated. Again, exception related information is initialised according to Section 5.3.

Finally, *non-compliance* and *external* events are manually triggered by a command available at the UI used to manage the workflow instances or, whenever possible, inside the UI tasks. When the command is issued, the exception handling workflow is instantiated with the running instance affected to the exception and the operator that signalled the event as responsible. The reaction time is initialised as relaxed since the user will be prompted to specify all exception related information using a dedicated UI as mentioned in Section 5.3.

### 5.6.3. Exception handling in the OSWF project

To implement the exception handling workflow of Figure 5.4, a model was developed in a XML file. The model is listed in Appendix E. The UIs allowing users to escalate an exception, to affect instances and to edit the description were built using Java Server Pages (JSP) programming to run over a Web environment. Their execution will be explained in Section 5.7 during the discussion of one user interaction example. In this section we will explain how the solution implements the recovery actions listed in Section 5.4. Since the consistency check is not yet implemented in OSWF, some stronger restrictions were made to inform the user on the consistency of the intervention.

The changes in the workflow models of the OSWF project are accomplished by editing the XML model files. A special method was developed to change a workflow model used by a particular workflow instance. This method will be used on various operations and changes the field NAME in the OS_WFENTRY table (cf. Section 5.6.1). A log entry is also generated for this operation. The description of the versioning system used to manage the models used by the instances is out of the scope of this work.

For the action *suspend/reinitialize instances*, the field STATE of the workflow OS_WFENTRY table is used. The `suspended` value on this field indicates that the workflow instance cannot start any activity. If a task started before the instance changes to the suspended state, a step transition can take place. The system should send messages to the person(s) executing manual tasks and to the supervisors of the automatic ones. The same process is adopted for the *abort instance* action but in this case, the field STATE is change to `killed`.

To implement *forward and backward jumps*, a new action is inserted in every step that uses as a parameter the number of the destination step. To identify whether it is backward jump or a forward jump the OS_HISTORYSTEP table is verified. If the destination step is in the table for this instance it is a backward jump otherwise the presence of a forward jump must be investigated.

To verify the consistency of a *backward jump*, the subnet as defined in Section 5.4 is identified. As this version of OSWF does not verify models consistency, backward jumps will be restricted to steps where the subnet only implements the sequence pattern as defined in [van der Aalst et al., 2002]. Later versions may implement the functionality as described in Section 5.4.

To investigate the presence of a *forward jump*, a simple algorithm is used to generate a tree of reachable steps from the current position. Once the destination step is found a forward jump presence is detected. Any loop is iterated only once. For complex models a depth limit can be defined. If the step is reachable the forward jump consistency is determined by a process similar to backward jumps. If the step is not reachable, the user is informed that this is a *jump* operation. Again, as in backward jumps only jumps in sequence patterns will be consistent. If the user wants to implement a *forward jump* with parallel execution of the tasks between the actual step and the destination the model must be changed. An AND-Split is inserted on the actual step and a task must be selected to synchronize the two parallel threads. An AND-Join is inserted on the task.

The *jump* operation requires a reachability test as mentioned in Section 5.4. The new step numbers are tested as in the forward jump above to investigate reachability. The jump is consistent if and only if the steps are reachable. Notice that in the jump operation, the instance may have more than one thread active. A new step number must be issued for every thread and the reachability study is made for all steps. The system is currently limited to change the value for two threads to bind the complexity of the generated tree of reachable states.

The *move operation* requires the edition of the model file by a workflow modeller to change the position of the task or block of tasks. Again, as the check for the consistency property is not yet implemented, this version will only allow moving blocks that implement the sequence pattern and that are moved within the limits of the same thread. The thread containing the moved block may only implement the sequence pattern.

For the *ad hoc refinement tool*, the list of the standard actions is defined in a dedicated XML file. Some minor adjustments had to be made to the OSWF project to assure the execution of these actions within the scope of the instance. The growth of this general purpose model is assured by a system designer that evaluates the activities executed by the users in every exception handling procedure and identifies the actions that should be included. For the actions defined in the model of the running instance no special code was developed. The actions are listed to the user that can select the desired one. All inserted tasks are executed on a new thread and join at the end of the model. Every model has a dummy join at the end inserted to join all these threads. This implementation simplifies the execution of these tasks.

In the alternative path of the *ad hoc extension* the user chooses another workflow model from a list with a predefined new trajectory for the remaining steps. As described in Section 5.4, two situations are considered: 1) the new model starts from the beginning with only one instance; and 2) all threads are transferred to a particular step in the new model. In the former case, since we assume the new model is consistent, the operation is consistent. In later case, the user indicates the step numbers for all threads in the new model and a reachability test is performed. According to the limits imposed on the reachability test mentioned above, it is not possible to transfer instances with more than two threads active. If the user still wants to transfer the instance, the system will assume inconsistency.

When small changes are applied to the model according to the transformation rules described in Section 3.4.2, consistency is assured and no further action is required. If the user does not comply with the mentioned transformations the operation is marked as inconsistent. When the user wants to apply a new model and the model is not available, s/he contacts the workflow modeller to develop a new one.

To support users resolving inserted inconsistencies, the limit to the consistency check described above must be taken into consideration. Nevertheless, users may verify inconsistencies with system support and resolve them using unstructured activities support. Visual tools displaying the model and active threads may be helpful to fulfil this task.

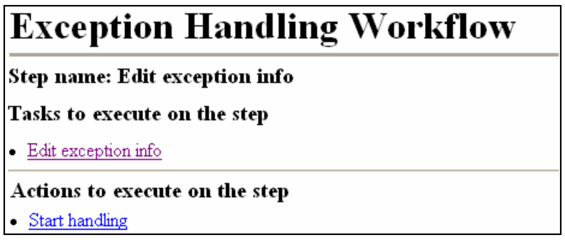
## 5.7 User interaction with the service

The exception handling workflow is instantiated according to the method described in Section 5.6.2 and may result from automatic or manual detection. In both cases one person is always associated to the exception and involved in the handling process. That person is either the one that manually detected the exception or someone involved in the workflow task that automatically generated an exception. Both will be naturally interacting with the system using available UI.

From the users' point of view, the handling process is managed through a Web page, which we designate Exception Handling Workflow (EHW). Figure 5.18 and 5.20 show the EHW user interface at two different states of the exception handling workflow model. To simplify the matching between the workflow model and the figures for the EHW user interface, the correspondence between workflow, Petri Net and OSWF terminologies is repeated. A workflow *state* is represented in Petri Nets by a *place* and corresponds to a *step* in OS. A workflow *task* is represented in Petri Nets by a *transition* and is spread through an *action* and the *tasks* available at the *step*. The OS *action* must be included because a workflow task is associated to state transition. (cf., Section 5.6.1., a *step* transition results from executing an OS *action*.) On the other hand, since the execution of task related activities is also included in the workflow task, the OS *step* where the activities are defined is also included.

The EHW page reports the current workflow state and manages the exception handling workflow. Figure 5.18 shows the step corresponding to place $P_1$ in Figure 5.4. In this step, only the task *Edit exception info* is active. If the *Edit exception info* task is executed, the exception information can be edited but no workflow transition will take place. To trigger state transition, the action named

*Start handling* must be executed. The user may then execute the task as many times as desired.


Exception Handling Workflow

Step name: Edit exception info

Tasks to execute on the step

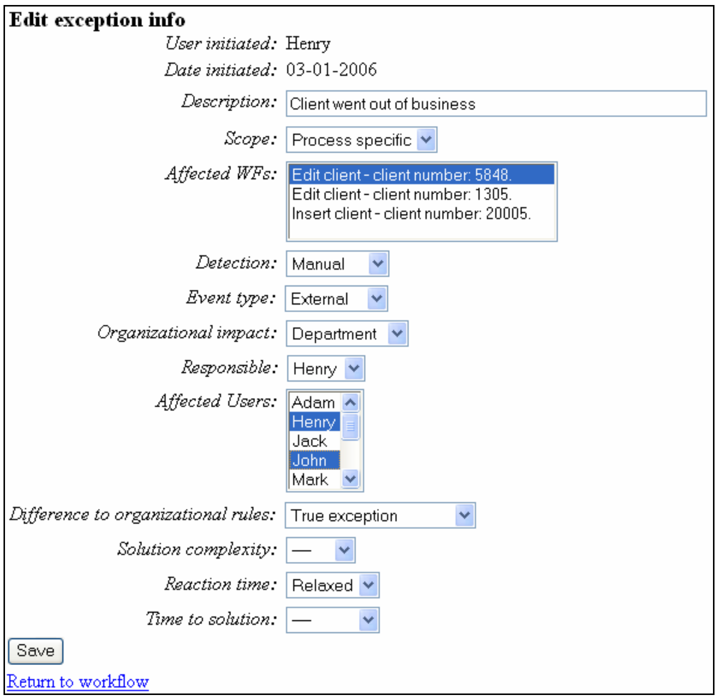
- Edit exception info

Actions to execute on the step

- Start handling

Figure 5.18. Exception handling workflow page (EHW)

In Figure 5.19 we show the details of the *Edit exception info* task, where the user defines the mandatory and optional diagnosis values discussed in Section 4.5. That person may also define a new responsible and a list of affected users and workflow instances. All affected users, including the person responsible, will be notified by the collaboration support component on the step transition. Users are notified using the API. The values in all task figures belong to an example that will be described in Section 6.1. They are only used here to exemplify the UI usage.

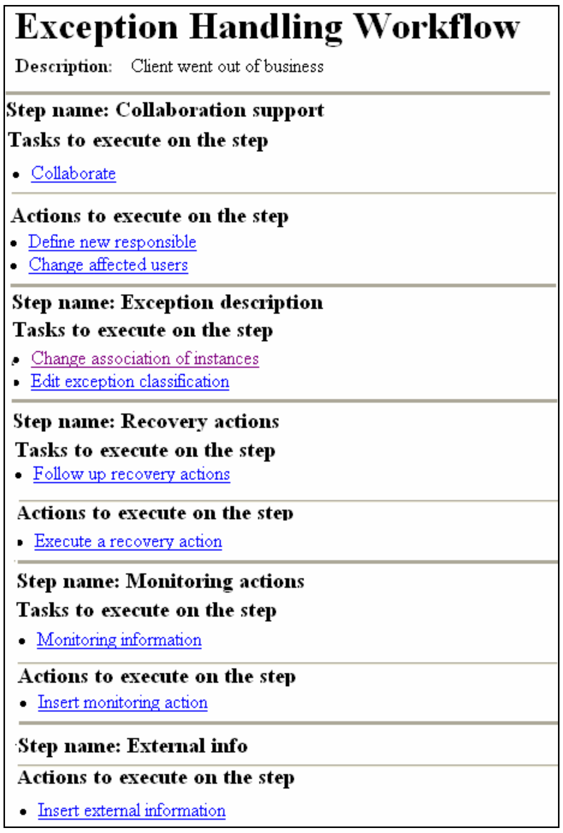


**Edit exception info**

| | |
|---|---|
| *User initiated:* | Henry |
| *Date initiated:* | 03-01-2006 |
| *Description:* | Client went out of business |
| *Scope:* | Process specific |
| *Affected WFs:* | Edit client - client number: 5848. Edit client - client number: 1305. Insert client - client number: 20005. |
| *Detection:* | Manual |
| *Event type:* | External |
| *Organizational impact:* | Department |
| *Responsible:* | Henry |
| *Affected Users:* | Adam, Henry, Jack, John, Mark |
| *Difference to organizational rules:* | True exception |
| *Solution complexity:* | — |
| *Reaction time:* | Relaxed |
| *Time to solution:* | — |

Figure 5.19. Editing the exception information

Considering again the EHW page, the *Start handling* action initiates the five parallel branches of the exception workflow model. Consequently, the EHW page will look like Figure 5.20. Observe that five steps are now available (corresponding to places $P_2$ through $P_6$ in Figure 5.4), allowing to collaborate with other persons involved, modify the exception description, execute recovery actions, execute monitoring actions or manage external information.



Exception Handling Workflow

Description: Client went out of business

Step name: Collaboration support

Tasks to execute on the step

- Collaborate

Actions to execute on the step

- Define new responsible
- Change affected users

Step name: Exception description

Tasks to execute on the step

- Change association of instances
- Edit exception classification

Step name: Recovery actions

Tasks to execute on the step

- Follow up recovery actions

Actions to execute on the step

- Execute a recovery action

Step name: Monitoring actions

Tasks to execute on the step

- Monitoring information

Actions to execute on the step

- Insert monitoring action

Step name: External info

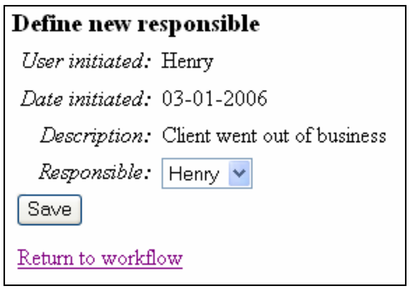
Actions to execute on the step

- Insert external information

Figure 5.20. EHW page handling the 5 parallel branches of the exception handling workflow

The *Collaboration support* step offers one task and two actions. The *collaborate* task can be synchronous or asynchronous and is implemented using a developed API. When asynchronous collaboration is selected, the system supports sending email messages between the persons handling the exceptional event. The generated email messages mixes information provided by the sender with information automatically generated by the collaboration component, which includes at least a link to the EHW page. Concerning the synchronous collaboration, the collaboration component supports instant messaging between the persons handling the exceptional event and interfaces with the exception history component to preserve the exchanged messages in context.

The *Define new responsible* action allows modifying the person responsible for the exception handling. This action is implemented by the Web page displayed in Figure 5.21, where the user may choose a new responsible by selecting a person from a combo box displaying the list of all persons that exist in the organization.

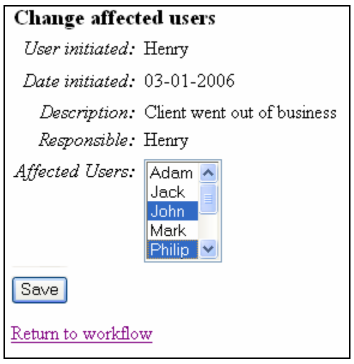


**Define new responsible**

| | |
|---|---|
| *User initiated:* | Henry |
| *Date initiated:* | 03-01-2006 |
| *Description:* | Client went out of business |
| *Responsible:* | Henry |

Figure 5.21. Choose a new responsible

The *Change affected users* action enables the selection of affected users, as shown by the Web page displayed in Figure 5.22. Note that the person responsible is not displayed because s/he is always affected by the exception handling process.

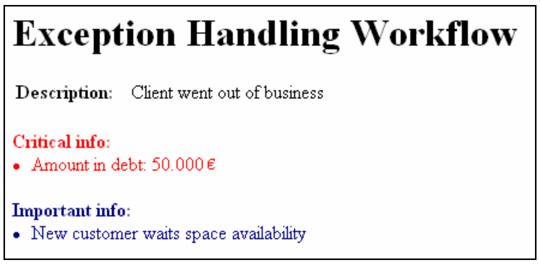


**Change affected users**

| | |
|---|---|
| *User initiated:* | Henry |
| *Date initiated:* | 03-01-2006 |
| *Description:* | Client went out of business |
| *Responsible:* | Henry |
| *Affected Users:* | Adam, Jack, John, Mark, Philip |

Figure 5.22. Change affected users

Concerning the *Edit exception classification* action in step *Exception description*, the Web page utilized to edit the exception classification is similar to the Edit exception info page shown in Figure 5.19 and is not shown. One additional functionality is that the user may share alert messages with attached files with the other persons involved. These alert messages may be classified as critical (displayed in red) or important (displayed in blue). Figure 5.23 illustrates how the alert messages are displayed in the EHW page. If none of these classifications is selected the message is only displayed inside the component. The Web page shown in Figure 5.24 enables changing the workflow affected instances.



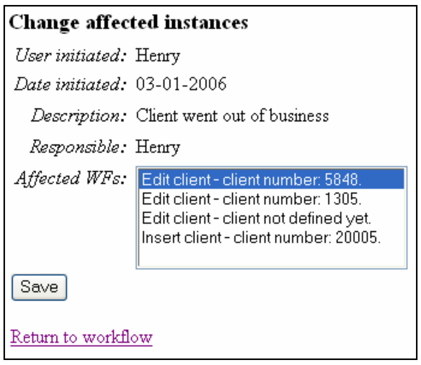
**Exception Handling Workflow**

**Description:** Client went out of business

**Critical info:**
- Amount in debt: 50.000 €

**Important info:**
- New customer waits space availability

Figure 5.23. EHW displaying alert messages at the top



**Change affected instances**

| | |
|---|---|
| *User initiated:* | Henry |
| *Date initiated:* | 03-01-2006 |
| *Description:* | Client went out of business |
| *Responsible:* | Henry |
| *Affected WFs:* | Edit client - client number: 5848.<br>Edit client - client number: 1305.<br>Edit client - client not defined yet.<br>Insert client - client number: 20005. |

Figure 5.24. Changing the affected instances

Concerning the *Recovery actions* step, the user must first select, among the affected instances, which ones to apply a recovery action. Then, one recovery action may be selected from the list discussed in Section 5.4. The implementation of these recovery actions requires low-level interventions in the OSWorkflow that will not be described in detail here.

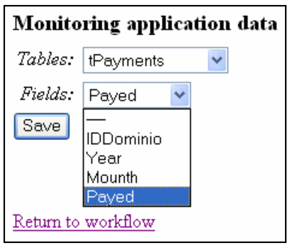
Regarding “Monitoring actions,” this *step* allows users storing relevant external information in the exception history. The user may select among the following information types: application data; workflow relevant data; workflow control data; links to Web resources; and text provided by users. Application data, workflow relevant data, and workflow control data follow the terminology defined by the WFMC [WfMC, 1999].

If application data resides on an accessible database, a reference can be inserted in the OSWorkflow configuration file to allow accessing the database. The monitoring action Web page then accesses the database metadata and displays the available tables and fields, so that the user may associate the monitoring action with a database field. In the Code listing 5.6, a XML component is used to provide the parameters to connect to a SQL Server. Another element not shown is then used to make all these parameters available to the application’s context.

```
<ResourceParams name="application-database">
  <parameter>
    <name>url</name>
    <value>
      jdbc:microsoft:sqlserver://comp:1433;Databasename=db
    </value>
  </parameter>
  <parameter>
    <name>driverClassName</name>
    <value>com.microsoft.jdbc.sqlserver.SQLServerDriver
    </value>
  </parameter>
</ResourceParams>
```

Code listing 5.6 XML component to link to application data

Figure 5.24 shows the Web page allowing users selecting a field from the `tPayments` table. The values obtained this way are always saved in the history component in textual format no matter their original type.



**Monitoring application data**

*Tables:* tPayments

*Fields:* Payed

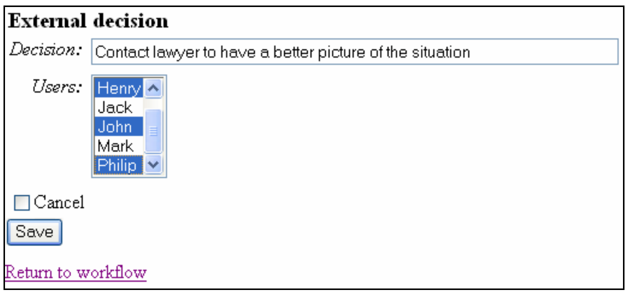
—
IDDominio
Year
Mounth
Payed

Save

Return to workflow

Figure 5.25. Selecting the table and field for a monitoring action

Finally, the "External info" *step* affords recording into the exception history any external information provided by users. Figure 5.26 displays the Web page recording information related with a decision. Note there is a field indicating if the decision is still valid.



**External decision**

*Decision:* Contact lawyer to have a better picture of the situation

*Users:* Henry, Jack, John, Mark, Philip

Cancel

Save

Return to workflow

Figure 5.26. Inserting external decision-making information

## 5.8 Exception data model

The information associated to every exception is represented in the data model of Figure 5.27. This model is highly integrated with the OSWorkflow data model to enable cross referencing. If the system is intended to be universal, an Application Programmer Interface specifying the primitives for data interchange, could be defined. As mentioned in Section 5.6.1, the `OS_WFENTRY` table stores all instances executed or being executed by the workflow engine. The table `tExceptions` stores the classification values for all dimensions identified in

Section 4.5 Table `tExcAffectedWFs` has the key identifiers of the `OS_WFENTRY` table and identifies the affected instances, while `tExcAffectedUsers` has the user names of the persons affected by the exception.

Table `tExceptions` stores old values for logging purposes. Every save operation on this table creates a new row with a reference to the old values. Child tables with the affected instances and users are duplicated and associated to the new rows.

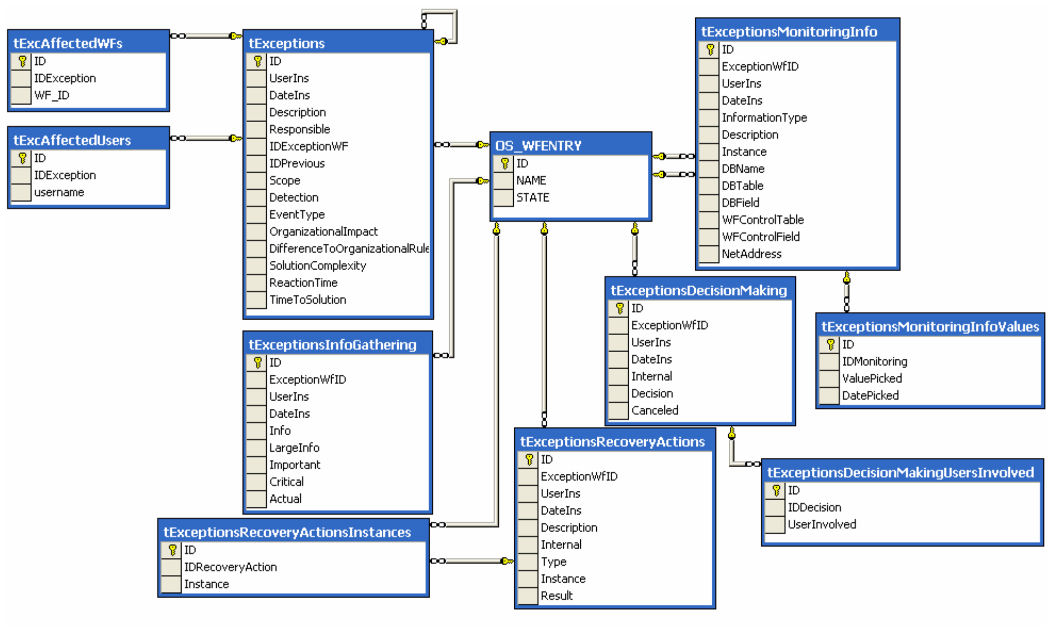


Figure 5.27. Exception referential model

Table `tExceptionsMonitoringInfo` stores data related with external monitoring information, e.g., if it concerns application data, workflow data or a link to an external site. The child table `tExceptionsMonitoringInfo` contains the data values. External decision-making information notified through interface E is stored in tables `tExceptionsDecisionMaking` and `tExceptionsDecisionMakingInvolvedUsers`. Recovery actions are kept by tables `tExceptionsRecoveryActions` and `tExceptionsRecoveryActionsInstances`. Finally, external information is stored in the `tExceptionsInfoGathering`.

The actual exception data model does not store all the exception related information identified in Section 4.5. In particular, the present version does not store affected tasks, data structures, data timers, model deviations, root cause and impact. This information has been identified as relevant for situation diagnosis but not critical for current implementations and therefore not yet implemented.

## 5.9 Summary

This chapter describes the solution architecture and implementation. We started by introducing our solution's architecture to implement the exception handling service. The architecture is based on our extended reference model and is composed by four components: *Exception Description*, *WF Interventions*, *Collaboration Support* and *Exception History*. The solution's interfaces with the WfMS and with the environment were also described.

A dedicated workflow implements the solution's components. Whenever an exception is detected, the exception handling workflow is instantiated. The workflow reflects the exception handling service procedure presented in the state diagram of Figure 4.1. On the other hand, the support to the basic problem functions identified in Section 4.3 is also assured. In particular, the intertwined play between diagnosis and recovery.

The mechanisms to automatically detect the exception types workflow, data, temporal and system/application are also described. In the case of manual exception detection, users instantiate the exception handling workflow.

A set of quasi-atomic recovery actions is described. These actions allow users to carry out the recovery procedures to bring the system back into a coherent state. The impact of the actions on instances' consistency is also discussed. Nevertheless, users are not restricted to implement the actions even if inconsistencies are inserted. They are advised and, if they decide to proceed with the action, the instance is marked as inconsistent.

Our solution is implemented using the OS open source project suite of components. The suite was explained to enable the description of the implementation of the solution's components by the dedicated workflow. Using example screens, the user interaction with the service was also described. Finally, the chapter finishes with the presentation of the exception data model that stores all the exception related information.

# Chapter 6

# Evaluation

As mentioned throughout this document, our proposed system has the objective to effectively support users on their reaction to unexpected exceptions. The effort placed on the system implementation reflects our concern with feasibility evaluation. We should consider, however, the concept developed in this thesis to deal with unexpected exceptions should also be verified in organizational scenarios. We have also placed some effort on this subject even though we recognize deeper research on the subject is still required.

Therefore, to validate our approach we have identified four relevant aspects:

1. Visioning – the approach used in our solution should be conceptually evaluated;

2. Feasibility evaluation – implement the solution;
3. Field tests – use the solution in case studies to verify its impact;
4. Usage – verify if the solution is adopted by organizations.

We have used the 9/11 example throughout this thesis as conceptually inspiring our solution. The example is further discussed in Section 6.1.

The solution implementation is described in this thesis. The developed code will be made available for the open source community and its usage will be followed up. Some work will also be carried out to integrate the solution with the OS framework to facilitate its acceptance.

We have been able to test our solution in a concrete situation: a Port Authority. During system tests, we were able to follow an exception since it was detected until it got solved. In Section 6.2. we discuss this event.

In Section 6.3 we discuss the usage of our exceptions characterization by a Brazilian company. Finally, our publications have also been referenced by various authors indicating that this area of research is still a hot topic. In Appendix D we list the publications that cited our work.

## 6.1 The 9/11 conceptually inspiring event

This case has been introduced and discussed throughout this dissertation. In this section we discuss how this thesis solution could have been used to support air traffic controllers during the 9/11 event. In order to be effective, we believe that the unexpected exception handling service should be integrated in the work space of the air traffic controllers. Therefore, we have connected our architecture with the WfMS and with the environment. On the other hand, the solution should also be integrated into the work environment so users interact with it in the sane way as they interact with the systems they usually work with. We will assume that such a system exists and is widely used by all air traffic controllers within the USA. We will now discuss how our solution could have been used.

As soon as the Boston air traffic controller looses the transponder signal from the screen, he signals an expected exception. He assumes that this is due to some temporary malfunctioning equipment. When he hears the voices in the cockpit saying “we’ve got some planes” he instantiates a new unexpected exception involving his supervisor and the control centre in Herndon with high priority. His supervisor involves someone else from the Boston control centre to hear the conversation between the air traffic controller and the pilot. The air traffic controller keeps following the plane using the radar screen. When he realizes that the plane is entering the New York control centre area, he involves his colleagues in the exception handling effort. When New York controllers connect to the exception handling service they will be informed on the situation details.

When the plane hits the tower, the New York controller issues the information on the system. A message could have been broadcasted to every controller to report any plane with no transponder signals. Using the system, the information could have been spread easier through the air traffic control system involving the most adequate operators on the handling effort.

A new monitoring task could have been inserted to follow all missing planes (to replace the white board). Controllers would report on a screen available to everyone on the suspicious flights. The screen would display flight information, the reporting control centre and the controller that has responsibility on the flight. Any update on the flight information could be issued as soon as available facilitating information flow. Table 6.1 is chronological list of the events and how users might have used the system.

Table 6.1. Timeline for the 9/11 events and system usage

| Time | Event | System usage |
|---|---|---|
| | American Airlines (AA) Flight 11 instance created at check in. | |
| 8:15 | AA Flight 11 transponder signals disappear and pilots do not answer air traffic controller calls. | Expected exception signalled. The recovery procedure is well known and the controller starts processing it. |
| 8:16 | Air traffic controller hears a voice in the cockpit: "We have some planes. Just stay quite and you will be ok." | Unexpected exception signalled. The situation is similar to a hijacked situation; however the phrase heard is not common and worries the national operations manager at the Herndon control centre. |
| 8:-- | AA flight 11 enters the New York control centre | Involve the New York air traffic controllers in the exception handling process. |
| 8:46 | AA flight 11 hits the north tower of the World Trade Center | Information inserted into the system. Everyone involved is aware of the situation.<br><br>Broadcast a message informing every controller in the USA air traffic system. Start the task to monitor every suspicious plane. |
| 9:03 | United Airlines (UA) Flight 175 hits the south tower. | Insert information into the system. |
| 9:03 to 9:07 | Air traffic control zero declared on the Northeast area (includes New York): clear the skies. | Broadcast the decision to inform every controller in USA to reroute planes. |
| 9:03 to 9:07 | Boston regions' air traffic control officials stop takeoffs and landings. | Broadcast the decision to inform every controller in USA to reroute planes. |
| 9:08 to 9:11 | Departures are stopped nationwide for aircraft heading to or through New York and Boston regions' airspace. | Broadcast the decision. |
| 9:25 | Federal Aviation Administration (FAA) stops takeoffs nationwide. | Insert information into the system and broadcast. |

| Time | Event | System usage |
|---|---|---|
| 9:35 | UA Flight 93 begins unauthorized climb, raising concerns it has been hijacked. | Insert information into the system |
| 9:38 | AA Flight 77 crashes into the Pentagon. | Insert information into the system |
| 9:45 | FAA orders all aircraft to land as soon as possible. | Broadcast the decision. Set collaborative mode in collaboration level strategy to improve local decision making |
| 10:06 | UA Flight 93 crashes in Shanksville, Pa. | Insert information into the system |

In a situation like the 9/11 event is difficult to conclude how people would have reacted if they had different tools at their disposal. Nevertheless, this scenario has inspired our work and helped us understanding how users may be supported on this type of events. We believe that a system like our solution could have worked as a facilitator spreading information about the event through the involved users. The telephone lines wouldn’t be the only way used between users to communicate and there was always a screen where they could look for updated information on the event. It would be easier to involve new users because they would have access to the information on the screen and be informed as new information was being collected.

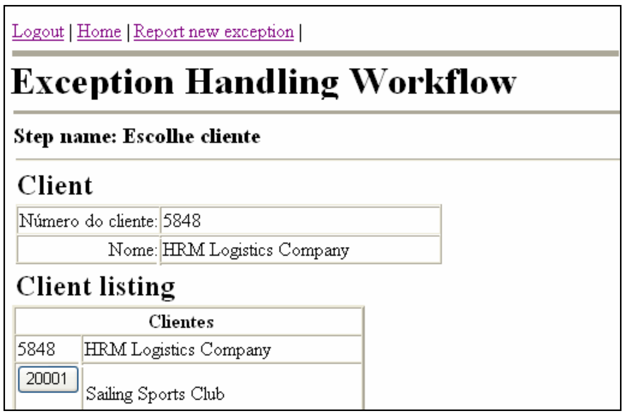
This example is relevant as a scenario where the purposed solution may adjust to a concrete situation users’ face on their organizational activities: working under model guidance and change to map guidance when the situation is no longer predicted in the model. Map guidance support to users was also illustrated by the usage of an interactive tool where users insert data about the suspicious planes. This information could have been constantly updated by the controllers and displayed on a large screen in the Herndon control centre.

## 6.2 The Port Authority Case

The Port Authority has the responsibility to manage all business activities within its jurisdiction that includes the river and the shore side. The Port Authority is a private company where the Portuguese state is the only shareholder and manages all vessels and cargo transfers to and from ships. All commercial activities installed on the shore are also under the jurisdiction of the Port Authority that issues licences and contracts for the rented places.

The businesses processes modelled in the Port Authority refer to the activity that manages space rentals within the Port Authority jurisdiction. Companies and individuals rent spaces for business activities for which they pay a fixed amount in a regular basis (e.g., some clients pay monthly while others pay yearly). For each rented space, there is a contract between the client and the Port Authority expressing all the conditions governing the business agreement, e.g., the description of the rented space, the time period that the space is rented, the payment periodicity and the amount to pay. A department with 10 employees negotiates all contracts, manages client related information and assures clients pay on time. These administrative processes were modelled using the OS platform. The modelled processes considered contractual activities were for new contracts, to change existing ones and to terminate them. At the end of every month, the system automatically instantiates a new process for every rented space that is supposed to pay its fee. A list of debts and free/occupied zones must be generated at any moment. Client related information is also managed by an appropriate process. On every implemented process, users have a link that enables initiating the exception handling process. To describe the exceptional event, user names were changed to preserve anonymity.

In the following we will describe the handling of a concrete exceptional event. Assume that Henry is updating the client's information when he is informed that the client has bankrupted. Figure 6.1 shows the Web page of the application workflow for the client editing task, where the link to manually signal an exception is shown at the top. On every task the user can instantiate a new exception.



Logout | Home | Report new exception |

Exception Handling Workflow

Step name: Escolhe cliente

Client

|  |  |
|---|---|
| Número do cliente: | 5848 |
| Nome: | HRM Logistics Company |

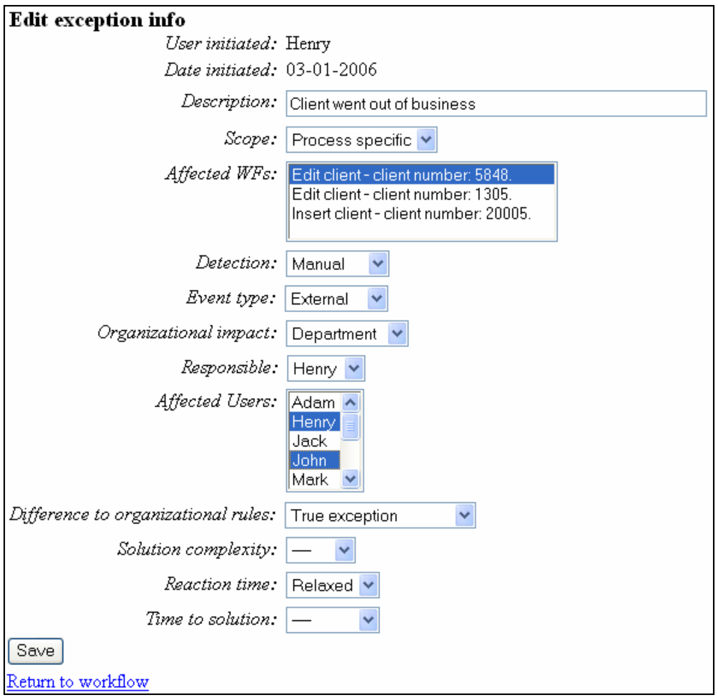
Client listing

| Clientes | |
|---|---|
| 5848 | HRM Logistics Company |
| 20001 | Sailing Sports Club |

Figure 6.1. Web page for the client edit workflow

After selecting this link, the user is prompted with the EHW page shown in Figure 5.18. From there, the exception classification must be accomplished, as shown in Figure 6.2[12]. Henry realizes that time is not critical and classifies it as relaxed. He also affects John, his direct supervisor, to the exception handling process. He does not define John as responsible because he wants to talk with him first. He inserts a brief exception description and classifies the exception as an external event with departmental impact. He also defines the exception as a true exception, since it never happened before. The dimensions scope, affected instances, and responsible were automatically defined by the system.

[12] The figure is repeated from page 182 to facilitate the reading



**Edit exception info**

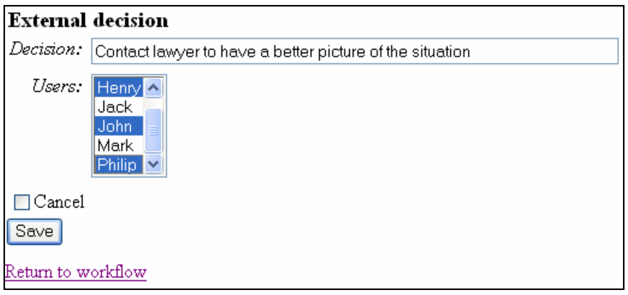
| | |
|---|---|
| *User initiated:* | Henry |
| *Date initiated:* | 03-01-2006 |
| *Description:* | Client went out of business |
| *Scope:* | Process specific |
| *Affected WFs:* | Edit client - client number: 5848. (selected); Edit client - client number: 1305.; Insert client - client number: 20005. |
| *Detection:* | Manual |
| *Event type:* | External |
| *Organizational impact:* | Department |
| *Responsible:* | Henry |
| *Affected Users:* | Adam; Henry (selected); Jack; John (selected); Mark |
| *Difference to organizational rules:* | True exception |
| *Solution complexity:* | — |
| *Reaction time:* | Relaxed |
| *Time to solution:* | — |

Figure 6.2. Exception related information

By following the *Start handling* link shown in Figure 5.18, Henry starts handling the exception. An email is generated to John with the exception handling information inserted by Henry and a link to the EHW shown in Figure 5.20.

John may then look at the situation in the EHW page and start a collaboration *task* with Henry. He decides using instant messaging. During the conversation, John realizes that the space occupied by the company is being requested by another company. He also recognizes that the client's debt is 50.000€. John tells Henry to insert this alert in the EHW (cf. Figure 5.23) and then involves Philip, from the lawyer department, in the exception handling process. John also decides to insert a monitoring task to identify whether the client has any other debts.

Philip is informed about the situation by email. After reading the email message, he decides to phone Henry to discuss the details. During the phone conversation, they decide that Philip will consult an external expert. Philip inserts a comment about this decision in the external information UI shown in Figure 6.3[13]. Henry will wait for any news.



**External decision**

*Decision:* Contact lawyer to have a better picture of the situation

*Users:* Henry, Jack, John, Mark, Philip

Figure 6.3. Inserting external decision-making information

Philip finds out from the expert that the Port Authority should notify the client by standard mail, giving 5 days to pay the debt. Obtaining no response, they should start a lawsuit action. Philip writes a letter draft and attaches it to the workflow as an entry message in the "edit exception classification" *action*. He then uses the "collaboration support" *step*, to discuss with Henry and John to decide on who will send the letter and who will follow this external action. The asynchronous email mechanism is adopted for that purpose.

Henry will be in charge of this external recovery action. John will also monitor the evolution of the case in order to decide or not to release the space to another client. If Henry finds out the company pays the older debts they have to reanalyze the situation. Again, Philip and John are notified about the new events. They realize the older debt does not allow them to start a law suit; however they decide that John should continuing monitoring this client. If the client pays all his old debts they close the exception handling process.

[13] The figure is repeated from page 187 to facilitate the reading

The system managed the interactions among users to handle this particular case. It was easy to involve an expert from another department in the handling process. Relevant decisions and event related information were easily spread through the involved users improving their knowledge about the situation details and their evolution. The relevant information related to the situation was also attached to the event so it can be used in future events.

## 6.3 Example Usage in a Brazilian Company

Our work has been tested by Nextsourcing[14], a Brazilian consultants company. In particular, the approach to event classification has been very useful in a big project relating information technology management in a Brazilian bank that has over 2.000 branches, 1.000 bank counters, 16.000 cash withdrawal machines and 4.000 servers. The project objective is to standardise the different helpdesks from the various technological subsystems, to plan technological upgrades and to develop a common business approach. The company used one paper published in a conference [Mourão and Antunes, 2003a].

## 6.4 Summary

In this chapter, we have established the solution's evaluation in four stages: 1) visioning; 2) feasibility evaluation; 3) field tests; 4) solution's usage.

The 9/11 example used throughout the thesis was used as conceptually inspiring our solution development. The usage of the solution on this scenario is also

[14] The company site is available at http://www.nextsourcing.com.br.

discussed. The Port Authority case study is a field study of our solution and the results were presented. The solution helped users handling this scenario. Nevertheless, we recognize that this topic still requires deeper investigation and more case studies should be carried out to understand the solution applicability to concrete scenarios and the organization types that take better advantages on the solution. However, the limited time associated to this work prevented deeper investigation on this subject.

It is also important to realize that part of the work has been already used by a private company in a large project involving a Bank, and that the published material has been cited by the researchers in the field.

# Chapter 7

# Conclusions and Future Work

The major concern addressed by this thesis is adjusting WfMS to real organizational scenarios. We discussed the current WfMS limitations and identified that users have the necessity to work freely in order to react to solicitations they face on their organizational activities. The processes carried out by organizations have been identified as belonging to a continuum from unstructured to structured behaviour. The majority of available systems support users only on both limits of the spectrum boundaries leaving a gap in the middle. Our solution supports users on both behaviours: works under model guidance (structured activities) and is capable of switching to map guidance support (unstructured activities) when necessary.

We started by characterizing the situations that force users to change from model guidance to map guidance. Existing taxonomies distinguish application failures, system failures, expected exceptions and unexpected exceptions. We have enriched this taxonomy by defining a continuum from expected exceptions to unexpected exceptions where we identify three classes: 1) true expected exceptions, if the event is equal to a known event, it is said to be truly expected and the organization has procedures to handle it; 2) extended expected exceptions, when the event is similar to a known one, even though not be entirely equal, and the handling procedure is applicable with some minor modifications; 3) effective unexpected exceptions, are situations for which the organization has no knowledge that may be used during the event handling. User involvement in the handling procedure increases when we move from the expected limits to the unexpected limits of the spectrum, because the existing organizational knowledge about the event decreases. If there is no knowledge about the event, the system can not be prepared to handle it and user involvement is mandatory. We have introduced another dimension on the proposed classification to distinguish the planning capacity the organization has on the reaction to the event. From the Organizational Sciences perspective it is recognized that the planning capacity depends on the uncertainty associated to the task. The higher uncertainty, the lower is the planning capacity. When users can not define a reaction plan, they must solve the situation on a problem solving basis – unstructured activities. These are the ad hoc exceptions, as opposed to the planned exceptions, where a plan can be defined before the handling procedure is initiated. Our solution is designed to handle *ad hoc effective unexpected exceptions* (referred as unexpected exceptions).

The proposed solution works under model guidance and changes to map guidance when an unexpected exception is detected. To be capable of supporting both behaviours the system should complement robustness with flexibility. This system's facet inspired us to define WfMS resilience as integrating both characteristics. We have characterized existing solutions to increase WfMS resilience according in five resilience levels: 1) systemic approaches to handle failures; 2) systemic approaches to handle expected exceptions; 3) restricted humanistic open-point approaches; 4) restricted humanistic metamodel approaches; and 5) unrestricted humanistic support for unstructured activities. Only one system in the related literature supports resilience level 5 (cf. Section 3.3 for a discussion on this subject). In conclusion, there is an unexplored field in WfMS resilience. We have then designed the extended reference model for a system that supports consistency level 5.

Therefore, we have focused on the level 5 characteristics of the solution to support unstructured activities. Two fundamental system requirements were established from the beginning: 1) users should not be restricted in any way by any system condition because they should have the flexibility they have when working without any system support – the completeness requirement; 2) the map guidance characteristics of unstructured activities require users to be fed with valuable updated information from the system and environment – the openness requirement. These are two mandatory system requirements to effectively support unstructured activities. On the other hand, some other characteristics were derived from the problem solving characteristics of the exception handling activities: 1) handling the event is a collaborative effort where key users must be involved; 2) involved users should be supported by collaborative mechanisms on their effort of diagnosing the situation and deciding on the most adequate recovery actions; 4) coordination is relaxed during unstructured activities and users must be supported on their coordination efforts; 5) the diagnosis is not finished on the first approach and is refined during the handling procedure, where the information may be collected my inserted monitoring tasks or any other means users have at their disposal; 6) decision support tools, designed for the concrete application scenario, should be integrated in the system to support users deciding the most adequate recovery actions to implement; and 7) users may need to consult the event history. The system functionality was established based on the identified system characteristics: escalation; monitoring; diagnosis; communication; collaboration; recovery; coordination; tools to determine the best solution; and history log. The organizational trajectory of the event and the escalation mechanism to propagate the event within the organization were then discussed.

The users' exception handling basic functions were indentified as: detection; diagnosing; monitoring; and handling. The intertwined play between diagnosis and monitoring/recovery is an important characteristic of the solution. The solution's architecture is derived from the extended reference model where the system components and interfaces are identified. The solution is implemented by a dedicated workflow model that reflects the mentioned characteristics. The solution was developed and runs on the OS WfMS. Implementation details were also discussed.

The solution proposed by this thesis is based on new concepts that enlarge the WfMS applicability in the organizational process spectrum. It is a new concept that has to be tested on different organizational scenarios using the case study methodology. We have used an effective unexpected exception event, the 9/11, as conceptually inspiring of our solution. We recognize the limitations of the example as a discussion about a radical event where it is difficult to devise how users could have reacted if some different technology was available. However, our solution is designed to support users on such "abnormal" scenarios and any other example would probably suffer from the same difficulty. This example was chosen because it is well documented and it is a truly effective unexpected exception. We have also validated the feasibility of the solution by implementing it using an open source platform, where we intend to publish our code to facilitate its usage by the community. We have also carried out field studies in a real case study. Nevertheless, we believe it is still necessary to conduct many other field studies to understand how the solution supports users in real organizational scenarios. It is also important to understand the type of organizations that have the major benefit from applying the solution. However, these studies require a long time frame to enable any results that are of real usage. We have place a significant effort on evaluating the solution considering the limited time frame we have available for this work. It is also important to realize that the final test to our solution will be obtained from the acceptance it will deserve from organizations and the amount of implementations.

Some implementation details should deserve attention in the near future. We believe that our system may be improved to allow the definition of multiple subgroups defined within the scope of a particular exception handling procedure. The subgroups may have specific sub-goals and be able to implement specific recovery procedures. Different collaboration mechanisms would be implemented at the subgroup and at the group levels. However, we have not implemented this issue because it does not invalidate our general assumption, even though, it may improve the solution support capability in some scenarios.

On the other hand, improving the solution interface with existing computer supported communication and collaboration tools may facilitate its usage and improve its acceptance.  The exception data model may also be completed to reflect all the items mentioned in Section 4.5 improving the solutions capability to describe the event. Finally, implementing the consistency check within OS is an important feature that improves the system support on removing inconsistencies and checking the consistency associated to the recovery actions implemented by users. The limits imposed to the reachability test could also be subjected to further research.

# References

Abbott, K. R. and Sarin, S. K., Experiences with workflow management: issues for the next generation, in *Proceedings of the 1994 ACM conference on Computer Supported Cooperative Work,* (Chapel Hill, North Carolina, United States, 1994), ACM Press,113-120.

Adams, M., *Facilitating Dynamic Flexibility and Exception Handling for Workflows*. PhD Thesis, Faculty of Information Technology, Queensland University of Technology, 2007.

Adams, M., Hofstede, A. H. T., Edmond, D., and van der Aalst, W., Facilitating Flexibility and Dynamic Exception Handling in Workflows, in *Proceedings of the CAiSE'05 Forum,* (Porto, Portugal, 2005),45-50.

Adams, M., Hofstede, A. H. T., and van der Aalst, W., Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows, in *On the Move to Meaningful Internet Systems 2006, OTM Confederated International Conferences,* (, 2006), Springer-Verlag,291-398.

Agostini, A. and De Michelis, G. Improving Flexibility of Workflow Management Systems. In van der Aalst, W. D. J. Oberweis, ed. *Business Process Management: Models, Techniques, and Empirical Studies*. Springer-Verlag, 2000a: 218-234.

Agostini, A. and De Michelis, G., A light workflow management system using simple process models, *Computer Supported Cooperative Work*, 9, 3 (2000b), 335-363.

Agostini, A., De Michelis, G., and Loregian, M., Undo in Workflow Management Systems, in *Business Process Management 2003*, (Eindhoven, The Netherlands, 2003), Springer-Verlag,321-335.

Alonso, G., Agrawal, D., and El Abbadi, A., Process synchronization in workflow management systems, in *Parallel and Distributed Processing, 1996. Eighth IEEE Symposium on*, (, 1996c),581-588.

Alonso, G., Agrawal, D., El Abbadi, A., and Kamath, M., Advanced Transaction Models in Workflow Contexts, in *12th International Conference on Data Engineering*, (New Orleans, Louisiana, 1996b), IEEE International,574-581.

Alonso, G., Agrawal, D., El Abbadi, A., Kamath, M., Guenthoer, R., and Mohan, C., Advanced Transaction Models in Workflow Contexts, in *Proc. 12th International Conference on Data Engineering*, (New Orleans, USA, 1996a).

Alonso, G., Hagen, C., Agrawal, D., El Abbadi, A., and Mohan, C., Enhancing the fault tolerance of workflow management systems, *IEEE Concurrency*, 8, 3 (2000), 74 -81.

Alonso, G., Kamath, M., Agrawal, D., El Abbadi, A., Guenthoer, R., and Mohan, C. *Failure Handling in Large Scale Workflow Management Systems*, 1994.

Bassil, S., Rinderle, S., Keller, R., Kropf, P., and Reichert, M., Preserving the Context of Interrupted Business Process Activities, in *7th International Conference on Enterprise Information Systems (ICEIS 2005)*, (Miami, USA, 2005).

Basten, T., *Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD Thesis, Eindhoven University of Technolog, 1998.

Bernstein, A., How can cooperative work tools support dynamic group process? bridging the specificity frontier, in *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, (Philadelphia, 2000), ACM Press,279-288.

Blumenthal, R. and Nutt, G. J., Supporting unstructured workflow activities in the Bramble ICN system, in *Proceedings of conference on Organizational computing systems*, (Milpitas, California, United States, 1995), ACM Press,130-137.

Borgida, A. and Murata, T., Tolerating Exceptions in Workflows: a Unified Framework for Data and Processes, in *Wacc '99*, (, 1999), ACM Press.

Bowers, J., Button, G., and Sharrock, W., Workflow From Within and Without: Technology and Cooperative Work on the Print Industry Shopfloor, in *European Conference on Computer Supported Cooperative Work*, (Stockholm, Sweden, 1995),51-66.

Breitbart, Y., Deacon, A., Schek, H. J., Sheth, A. P., and Weikum, G., Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows, *ACM SIGMOD Record*, 22, 3 (1993), 23-30.

Burns, T. and Stalker, G., *The management of innovation*. London: Tavistock Publications, 1961.

Bussler, C., Enterprise wide workflow management, *IEEE Concurrency*, 7, 3 (1999), 32-43.

Casati, F., *Models, Semantics, and Formal Methods for the Design of Workflows and their Exceptions*. PhD Thesis, Politecnico di Milano, 1998.

Casati, F., Ceri, S., Paraboschi, S., and Pozzi, G., Specification and Implementation of Exceptions in Workflow Management Systems, *ACM Transactions on Database Systems*, 24, 3 (1999), 405-451.

Casati, F., Ceri, S., Pernici, B., and Pozzi, G., Workflow Evolution, *Data and Knowledge Engineering*, 24, 3 (1996), 211-238.

Casati, F. and Pozzi, G., Modelling exceptional behaviors in commercial workflow management systems, in *Proc. IFCIS, International Conference on Cooperative Information Systems, CoopIS '99*, (Edinburgh, UK, 1999), IEEE International,127-138.

Chen, Q. and Dayal, U., A transactional nested process management system, in *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, (, 1996),566-573.

Chiu, D. K., *Exception Handling in an Object-oriented Workflow Management System*. PhD Thesis, Hong Kong University of Science and Technology, 2000.

Chiu, D. K., Li, Q., and Karlapalem, K., WEB Interface-Driven Cooperative Exception Handling in ADOME Workflow Management System, *Information Systems*, 26, 2 (2001), 93-120.

Combi, C., Daniel, F., and Pozzi, G., A Portable Approach to Exception Handling in Workflow Management Systems, in *OTM Conferences - CoopIS'06*, (2006), Springer-Verlag,201-218.

Dayal, U., Hsu, M., and Ladin, R., Organizing Long-Running Activities with Triggers and Transactions, in *Int. Conf. on Management of Data (SIGMOD'90),* (Atlantic City, NJ, USA, 1990).

Dayal, U., Hsu, M., and Ladin, R., A Transactional Model for Long-Running Activities, in *17th Int. Conf. on Very Large Data Bases (VLDB'91),* (Barcelona, Spain, 1991).

De Michelis, G. and Grasso, M. A., Situating conversations within the language/action perspective: the Milan conversation model, in *Proceedings of the 1994 ACM conference on Computer supported cooperative work,* (Chapel Hill, North Carolina, USA, 1994), ACM Press.

Deiters, W. and Gruhn, V., The Funsoft Net Approach to Software Process Management, *International Journal of Software Engineering and Knowledge Engineering*, 4, 2 (1994), 229-256.

Dellarocas, C. and Klein, M., A Knowledge-based approach for handling exceptions in business processes, in *Proc. of the 8th Workshop on Information Technologies and Systems (WITS'98),* (Helsinki, Finland, 1998).

Donaldson, L. The normal science of structural contingency theory. In Hardy, C. N. W., ed. *Handbook of organization studies*. Sage Publications, 1996: 57-76.

Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., and Zbyslaw, A., Freeflow: mediating between representation and action in workflow systems, in *Pro. of the 1996 ACM conference on Computer Supported Cooperative Work*, (New York, 1996), ACM Press.

Eder, J. and Liebhart, W., The Workflow Activity Model WAMO, in *Int. Conf. on Cooperative Information Systems*, (Vienna, Austria, 1995).

Eder, J. and Liebhart, W., Workflow Recovery, in *1st IFCIS Intl. Conf. on Cooperative Information Systems (CoopIS'96)*, (Brussels, Belgium, 1996), IEEE International,124 - 134.

Eder, J. and Liebhart, W., Contributions to Exception Handler in Workflow Management, in *Int. Conf. on Extended Database Technology (EDBT'98), Workshop on Workflow Management Systems*, (Valencia, Spain, 1998).

Edmond, D. and Hofstede, A. H. T., A Reflective Infrastructure for Workflow Adaptability, *Data and Knowledge Engineering,* 34, 3 (2000), 271-304.

Ellis, C., Information Control Nets: A mathematical model of office information flow, in *Proc. of the 1979 ACM conf. on Simulation and Modelling of Computer Systems*, (, 1979),225-239.

Ellis, C. and Keddara, K. A Workflow Change is a Workflow. In van der Aalst, W. D. J. Oberweis, ed. *Business Process Management: Models, Techniques, and Empirical Studies*. Springer-Verlag, 2000: 201-217.

Ellis, C., Keddara, K., and Rozenberg, G., Dynamic change within workflow systems, in *Proc. of conf. on Organizational computing systems*, (Milpitas, CA, USA, 1995), ACM Press,10-21.

Ellis, C., Keddara, K., and Wainer, J. Modeling Workflow Dynamic Changes Using Timed Hybrid Flow Nets. In van der Aalst, W. D. M. Giorgio Ellis, ed. *Workflow Management: Net-based Concepts, Models, Techniques, and Tools (WFM'98)*. Lisbon, Portugal, 1998: 109-128.

Ellis, C. and Nutt, G. J., Office Information Systems and Computer Science, *ACM Computing Surveys*, 12, 1 (1980), 27-60.

Ellis, C. and Nutt, G. J., Modeling and enactment of workflow systems, in *Application and Theory of Petri Nets*, (Chicago, Illinois, USA, 1993), Springer-Verlag,1-16.

Esparza, J. and Nielsen, M., Decibility Issues for Petri Nets - a survey, *Journal of Information Processing and Cybernetics*, 30, 3 (1994), 143-160.

Faustmann, G., Configuration for Adaptation - A Human-centered Approach to Flexible Workflow Enactment, *Computer Supported Cooperative Work*, 9, 3 (2000), 413-434.

Fayol, H., *General and Industrial Management*. London: Pitman, 1919.

Galbraith, J. R., *Organization Design*. Addisson-Wesley, 1977.

Georgakopoulos, D., Hornick, M., and Sheth, A. P., An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure, *Distributed and Parallel Databases*, 3, 2 (1995), 119-154.

Goodenough, J. B., Exception handling: issues and a proposed notation, *Communications Of The ACM*, 18, 2 (1975), 683-693.

Grigori, D., Casati, F., Dayal, U., and Shan, M. C., Improving Business Process Quality through Exception Understanding, Prediction, and Prevention, in *27th Int. Conf. on Very Large Databases (VLDB'01)*, (Rome, Italy, 2001).

Guimarães, N., Antunes, P., and Pereira, A. P. The Integration of Workflow Systems and Collaboration Tools. In Dogaç, A. K. Leonid Ozsu, ed. *Advances in Workflow Management Systems and Interoperability*. Istambul, 1997.

Hagen, C. and Alonso, G., Exception Handling in Workflow Systems, *IEEE Transactions On Software Engineering*, 26, 10 (2000), 943-958.

Hammer, M., Howe, W. G., Kruskal, V. J., and Wladawsky, I., A very high level programming language for data processing applications, *Communications Of The ACM*, 20, 11 (1977), 832-840.

Han, Y. *HOON - A Formalism Supporting Adaptive Workflows*, 1997.

Han, Y., Sheth, A. P., and Bussler, C., A Taxonomy of Adaptive Workflow Management, in *Conf. on Computer Supported Cooperative Work - Workshop - Towards Adaptive Workflow Systems*, (Seattle, WA, USA, 1998), ACM Press.

Hatch, M., *Organization Theory - modern, symboplic, and post-modern perspectives*. Oxford University Press, 2006.

Hayes, N., Work-arounds and Boundary Crossing in a High Tech Optronics Company: The Role of Co-operative Workflow Technologies, *Computer Supported Cooperative Work*, 9, 3 (2000), 435-455.

Heinl, P., Exceptions During Workflow Execution, in *Int. Conf. on Extended Database Technology (EDBT'98), Workshop on Workflow Management Systems,* (Valencia, Spain, 1998).

Hollingswoorth, D. *Workflow Management Coalition - The Reference Model TC00-1003*. WFMC, 1995.

Hsu, M. and Kleissner, K., ObjectFlow: Towards a process management infrastructure, *Distributed and Parallel Databases,* 2 (1996), 169-194.

Hwang, S. Y., Ho, S. F., and Tang, J., Mining Exception Instances to Facilitate Workflow Exception Handling, in *6th Int. Conf. on Database Systems for Advanced Applications*, (Hsinchu, Taiwan, 1999).

Jablonski, S. and Bussler, C., *Wokflow Management: Modeling Concepts, Architecture, and Implementation*. UK: Thompson Computer Press, 1996.

Jin, W. W., Rusinkiewicz, M., Ness, L., and Sheth, A. P., Concurrency control and recovery of multidatabase work flows in telecommunication applications, in *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, (Washington, D.C., United States, 1993), ACM Press,456-459.

Jorgensen, H. D., Interaction as Framework for Flexible Workflow Modelling, in *Group '01*, (Boulder, Colorado, USA, 2001), ACM Press.

Kamath, M. and Ramamritham, K., Failure Handling and Coordinated Execution of Concurrent Workflows, in *Proc. of 14th International Conference on Data Engineering*, (Orlando, Florida, 1998),334-341.

Kiepuszewski, B., Hofstede, A. H. T., and Bussler, C., On Structured Workflow Modelling, in *Twelfth International Conference on Advanced Information*

*Systems Engineering (CAiSE'2000)*, (Stockholm, Sweden, 2000), Springer-Verlag,431-445.

Kiepuszewski, B., Hofstede, A. H. T., and van der Aalst, W. *Fundamentals of Control Flow in Workflows*, 2001.

Klein, M. and Dellarocas, C., A Knowledge-Based Approach to Handling Exceptions in Workflow Systems, *Computer Supported Cooperative Work*, 9, 3 (2000), 399-412.

Leymann, F., Workflow-based applications, *IBM Systems Journal*, 36, 1 (1997), 102-123.

Luo, Z., *Knowledge sharing, Coordinated Exception Handling, and Intelligent Problem Solving for Cross-Organizational Business Processes*. PhD Thesis, Dep. of Computer Sciences, University of Georgia, 2001.

Luo, Z., Sheth, A. P., Kochut, K. J., and Arpinar, I. B. *Exception Handling for Conflict Resolution in Cross-Organizational Workflows*, 2002.

Medina-Mora, R., Winograd, T., Flores, R., and Flores, F., The action workflow approach to workflow management technology, in *Cscw'92*, (Toronto, Ontario, Canada, 1992), ACM Press,281-288.

Mintzberg, H., *Estrutura e Dinâmica das Organizações*. Publicações Dom Quixote, 1999.

Mohan, C., Alonso, G., Guenthoer, R., and Kamath, M., Exotica: A Research Perspective on Workflow Management Systems, *Data Engineering Bulletin*, 18, 1 (1995), 19-26.

Morgan, G., *Images of Organizations*. Sage Publications, 1997.

Mourão, H. R. and Antunes, P., Suporte à Intervenção de Operadores no Tratamento de Excepções em Fluxos de Trabalho, in *4ª Conferência da Associação Portuguesa de Sistemas de Informação*, (Porto, Portugal, 2003a),29-42.

Mourão, H. R. and Antunes, P., Supporting Direct User Interventions in Exception Handling in Workflow Management Systems, in *9th International Workshop on Groupware, CRIWG 2003*, (Autrans, France, 2003b), Springer-Verlag,159-167.

Mourão, H. R. and Antunes, P., Exception Handling Through a Workflow, in *CoopIS 2004: Proceedings of the conference: On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, (Agia Napa, Cyprus, 2004c), Springer-Verlag,37-54.

Nielsen, M., Rozenberg, G., and Thiagarajan, P., Elementary transition systems, *Theoretical Computer Science*, 16, 1 (1992), 3-33.

Nomura, T., Hayashi, K., Hazama, T., and Gudmundson, S., Interlocus: workspace configuration mechanisms for activity awareness, in *Conf. on Computer-Supported Cooperative Work*, (Seattle, Washington, USA, 1998), ACM Press.

Nutt, G. J., The evolution towards flexible workflow systems, *Distributed Systems Engineering Journal*, 3, 4 (1996), 176-294.

OpenSymphony (2007), The Open Symphony Project, www.opensymphony.com, last consulted on 2007/10/09

Perrow, C., *Complex organizations - a critical essay*. McGraw-Hill, 1986.

Perrow, C., *Normal Accidents - Living with High-Risk Systems*. Princeton University Press, 1999.

Petri, C. A., *Kommunikation mit Automaten*. PhD Thesis, Institut fur instrumentelle Mathematik, Bonn instrumentelle Mathematik, 1962.

Reichert, M. and Dadam, P., ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control, *Journal of Intelligent Information Systems*, 10, 2 (1998), 93-129.

Reichert, M., Dadam, P., and Bauer, T., Dealing with Forward and Backward Jumps in Workflow Management Systems, *Software and Systems Modeling*, 2, 1 (2003), 37-58.

Reisig, W., *Petri Nets: An Introduction*. Springer-Verlag, 1985.

Rinderle, S., *Schema Evolution in Process Management Systems*. PhD Thesis, University of Ulm, 2004b.

Rinderle, S., Reichert, M., and Dadam, P., Evaluation of Correctness Criteria for Dynamic Workflow Changes, in *Proc. Int'l Conf. on Business Process Management (BPM '03)*, (Eindhoven, Netherlands, 2003), Springer-Verlag,41-57.

Rinderle, S., Reichert, M., and Dadam, P., Correctness criteria for dynamic changes in workflow systems - a survey, *Data and Knowledge Engineering*, 50, 1 (2004a), 9-34.

Russel, N., van der Aalst, W., and Hofstede, A. H. T., Workflow Exception Patterns, in *18th International Conference on Advanced Information Systems Engineering (CAiSE'06)*, (, 2006), Springer-Verlag,288-302.

Saastamoinen, H., *On the Handling of Exceptions in Information Systems*. PhD Thesis, University of Jyväskylä, 1995.

Sadiq, S. W., Handling Dynamic Schema Change in Process Models, in *Database Conference, 2000. ADC 2000. Proceedings. 11th Australasian,* (, 2000a), IEEE International,120-126.

Sadiq, S. W., On Capturing Exceptions in Workflow Process Models, in *Proceedings of the 4th International Conference on Business Information Systems,* (Poznan, Poland, 2000c).

Sadiq, S. W., Marjanovic, O., and Orlawska, M., Managing Change and Time in Dynamic Workflow Processes, *International Journal of Cooperative Information Systems*, 9, 1 (2000b).

Schmidt, K., Of maps and scripts - the status of formal constructs in cooperative work, in *Proceedings of the international ACM SIGGROUP conference on Supporting group work (GROUP '97): the integration challenge,* (Phoenix, Arizona, United States, 1997), ACM Press,138-147.

Sheth, A. P., Georgakopoulos, D., Joosten, S. M., Rusinkiewicz, M., Scacchi, W., Wileden, J., and Wolf, A. L., Report from the NSF workshop on workflow and process automation in information systems, *ACM SIGMOD Record*, 25, 4 (1996), 55-67.

Strong, D. M. and Miller, S. M., Exceptions and Exception Handling in Computerized Information Systems, *ACM Transactions on Information Systems*, 13, 2 (1995).

Suchman, L. A., Office procedure as practical action: models of work and system design, *ACM Transactions on Office Information Systems*, 1, 4 (1983), 320-328.

Suchman, L. A., *Plans and Situated Actions*. MIT Press, 1987.

Suchman, L. A., Do Categories Have Politics? The Language/Action Perspective Reconsidered, in *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, (, 1993),1-14.

USA TODAY, 2007, Part I. Terror attacks brought drastic decision: Clear the skies. www.usatoday.com/news/sept11/2002-08-12-clearskies_x.htm, consulted on 2007/10/09

van der Aalst, W. Verification of Workflow Nets. In Azéma, P. B. G., ed. *Application and Theory of Petri Nets*. Berlin, Germany: Springer-Verlag, 1997: 407-426.

van der Aalst, W., The Application of Petri Nets to Workflow Management, *Journal of Circuits, Systems and Computers*, 8, 1 (1998), 21-66.

van der Aalst, W. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In van der Aalst, W. D. J. Oberweis, ed. *Business Process Management: Models, Techniques, and Empirical Studies*. Berlin: Springer-Verlag, 2000: 161-183.

van der Aalst, W., Exterminating the Dynamic Change Bug: A Concrete Approach to Support Change, *Information Systems Frontiers*, 3, 3 (2001), 297-317.

van der Aalst, W. and Basten, T., Inheritance of workflows: an approach to tackling problems related to change, *Theoretical Computer Science*, 270, 1 (2002), 125-203.

van der Aalst, W., Basten, T., Verbeek, H., Verkoulen, P., and Voorhoeve, M., Adaptive Workflow: On the interplay between flexibility and support, in *Proceedings of the First International Conference on Enterprise Information Systems,* (Setúbal, Portugal, 1999),353-360.

van der Aalst, W. and Berens, P., Beyond Workflow Management: Product-Driven Case Handling, in *GROUP 2001, Boulder, Colorado, USA*, (, 2001), ACM Press.

van der Aalst, W., Hofstede, A. H. T., Kiepuszewski, B., and Barros, A. *Workflow Patterns*, 2002.

van der Aalst, W. and van Hee, K., *Workflow Management*. London, England: MIT Press, 2002.

Vojevodina, D., Kulvietis, G., and Bindokas, P., The method for e-business exception handling, in *Intelligent Systems Design and Applications, 2005. ISDA '05. Proceedings. 5th International Conference on,* (, 2005), IEEE International,203-208.

Wachter, H., ConTracts: a means for improving reliability in distributed computing, in *Compcon Spring '91. Digest of Papers*, (, 1991),574-578.

Weber, B. and Wild, W., An Agile Approach to Workflow Management, in *Proceedings of Modellierung 2004*, (Marburg, Germany, 2004),187-201.

Weigand, H., Introduction to Special Issue on: Two decades of the language-action perspective, *Communications Of The ACM*, 49, 5 (2006), 44-46.

Weske, M., Formal foundation and conceptual design of dynamic adaptations in a workflow management system, in *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, (, 2001),2579-2588.

Winograd, T., Categories, disciplines, and social coordination, *Computer Supported Cooperative Work*, 3, 2 (1994), 191-197.

Winograd, T., Designing a new foundation for design, *Communications Of The ACM*, 49, 5 (2006), 71-74.

Winograd, T. and Flores, F., *Understanding Computers and Cognition: A new Foundation for Design*. Norwood, New Jersey: Ablex Pubs Corporation, 1986.

Worah, D. and Sheth, A. P. Transactions in Transactional Workflows. In Jajodia, S. K. Larry, ed. *Advanced Transaction Models and Architectures*. Kluwer Academic Publishers, 1997.

WfMC 1999, *Workflow Management Coalition - Terminology & Glossary, TC00-1011*. WFMC, 1999.

WfMC 2007, The Workflow Management Coalition, Http://www.wfmc.org, last consulted on 2007/10/09

Zacarias, M., Caetano, A., Pinto, S., and Tribolet, J. Modeling Contexts for Business Process Oriented Knowledge Support. In *Knowledge Management for Distributed Agile Processes*. Springer-Verlag, 2005b.

Zacarias, M., Marques, A., Pinto, S., and Tribolet, J., Enhancing Collaboration with Business Context, in *International Workshop on "Cooperative Systems and Context." 5th International and Interdisciplinary Conference on Modeling and Using Contexts*, (Paris, France, 2005a).

# Appendixes

## List of appendixes

# Appendix A – Using Petri Nets to model workflows

Petri Nets have firstly been introduced by Carl Adam Petri [1962]. Since then, Petri Nets have been applied to various areas and theoretically investigated. In fact, they are based on sound theoretical foundations and therefore are a natural choice for modelling systems behaviour [Reisig, 1985; Basten, 1998]. In particular, Petri Nets have been proposed for modelling workflows in WfMSs [van der Aalst, 1998; Ellis and Nutt, 1993; Saastamoinen, 1995].

We will abstract from introducing the formalism associated to Petri Nets algebra because it is not required by our approach. We describe Petri Nets informally to simplify the reading. The interested reader may consult Aalst [1998] for the formal representation of nets within the context of workflow systems. In this appendix, we will start by introducing Petri Nets and then explain how workflow models can be represented using these nets.

After establishing the static characteristics of Petri Nets we will concentrate on how they evolve over: the dynamic behaviour.

> *A Petri Net is defined by Aalst [van der Aalst, 1998] as a bipartite graph with two node types called place and transitions. The nodes are connected via direct arcs. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by line segments perpendicular to the arcs.*

Figure A.1 is a Petri Net with one start place $P_1$ and one sink place $P_6$. The start place is a place that only has outgoing arcs and a sink place is a place that only has incoming arcs. A place is said to connect to a transition if there is an outgoing arc from the place to the transition and a transition connects to a place on the same conditions, i.e., there is an outgoing arc from the transition to the place.

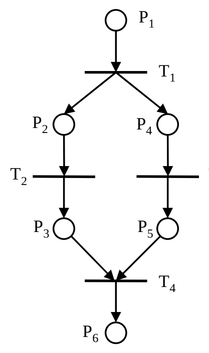


Figure A.1. Example of a Petri Net

At any time, the places may contain one or more tokens that are usually called marks and drawn as black dots. A transition is said to be enabled if every input place of that transition has at least one marking. If the transition is enabled, it may fire. The firing of a transition corresponds to remove one mark from every input place of the transition and placing one mark on every output place of the transition. Figure A.2 represents the firing of transition $T_1$ in the Petri Net of Figure A.1. The mark in place $P_1$ (Figure A.2.a) was removed and a mark was added to places $P_2$ and $P_4$ in Figure A.2.b).

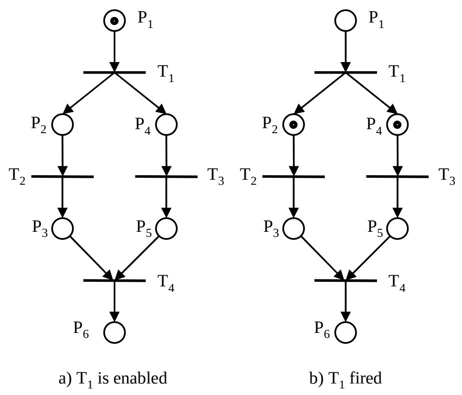


Figure A.2. Example of firing a transition in a Petri Net

To model workflows, transitions in Petri Nets correspond to tasks in workflows and places to conditions. A new mark in the input place $P_1$ of the Petri Net in

Figure A.1 corresponds to a new instance in the workflow model. Then, task $T_1$ can be executed if the condition associated to place $P_1$ evaluates to true.

# Appendix B – Consistency proof for forward and backward jumps

## Backward jump

The backward jump is consistent if the subnet starting at the destination place ($P_1$ in Figure B.1.a)) of the jump and finishing at the original place ($P_6$ in Figure B.1.a)) can be isolated (including every node in every branch leading from the start place to the end and every arc that finish or start on those nodes.) From now on this subnet will be the jump subnet. It is also assumed that the original net where the jump is performed is consistent.

*Proof*

To proof the above rule the fourth statement of the Theorem 3 in [van der Aalst, 2000] is used. The starting premise of the theorem is to compose a net $N_3$ by the replacement of a transition in a net $N_1$ by another net $N_2$. Some properties of the nets can be obtained from this composition. In particular, the fourth statement: $N_1$ and $N_2$ are consistent if and only if $N_3$ is consistent[15].

The first step is to proof that the delimited net is consistent. If the jump subnet is removed from the original net and a transition is inserted on its place the above

[15] The concept of sound found in the theorem is equivalent to consistency without the requirement that tasks can only be fired once for any achievable marking (safeness concept in the theorem.) In the presented version of the statement the concept of soundness was replaced by consistency and the mention to safeness has been removed since it is already included in this thesis definition of consistency. The original statement is: N1 and N2 are safe and sound if and only if N3 is safe and sound.

mentioned statement can be used. The obtained net will be referred as the final net in this transformation. It should be emphasised that this transformation is in the opposite direction of the Theorem (from the composed original net to two individual subnets.) Therefore the original net is $N_3$ in the Theorem, the jump subnet is $N_2$ and the final net is $N_1$. Then, from the mentioned statement, and assuming the original net is consistent, both the removed jump net and the final net are also consistent. Hence, the delimited jump subnet is consistent.

The proof proceeds by noting that jumping to a previous location is equivalent to insert a copy of the jump subnet just after the place where the jump is originated (Figure B.1.c)). A dummy transition and place are inserted after $P_6$ in the Figure B.1.a) obtaining Figure B.1.b) [16]. Then, the dummy transition is replaced by a copy of the jump subnet which results in the net in Figure B.1.c). It should be emphasised that jump subnet have been kept simple to reduce the figure complexity. However, the proof holds for any net that complies with the rule.

Since the insertion of a dummy transition in transformation $Trans_1$ is a trivial operation it is only necessary to proof that $Trans_2$ transforms a consistent net into another consistent net, i.e., the net obtained by replacing the dummy transition $T_d$ in the original net of Figure B.1.b) by the jump subnet is consistent. The original net corresponds to $N_1$ in the theorem, the jump subnet to $N_2$ and the obtained net in Figure B.1.c) to $N_3$. According to the fourth statement the final net is consistent if and only if both the original net and the jump subnet are also consistent. Since it is assumed that the original net is consistent and, as proofed above, the jump subnet is also consistent the final net is consistent. Hence the backward jump is consistent.

[16] Inserting a dummy transition and a place does not alter the net properties. The dummy transition is called a silent action since it is not observable. Refer to [van der Aalst and Basten, 2002] on this issue.

□

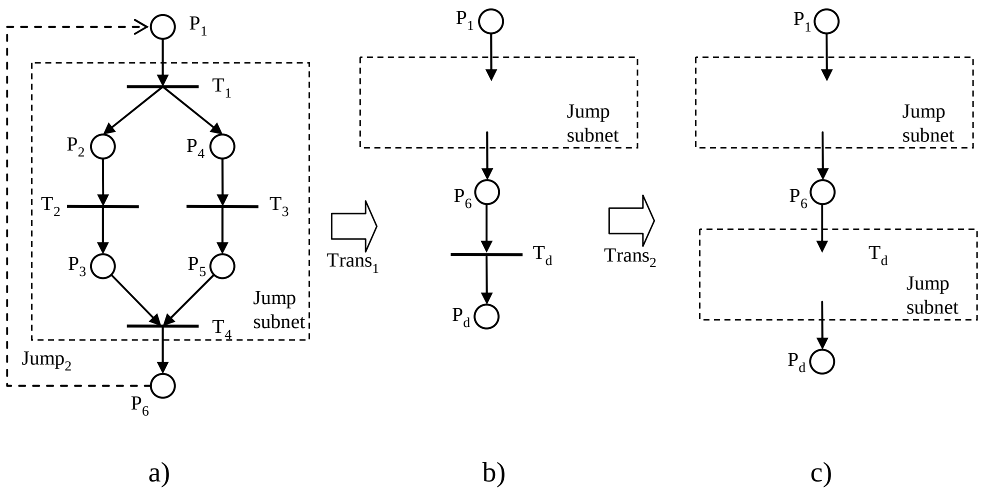


Figure B.1. Equivalent model for the backward jump with parallel execution

## Forward jump by skipping tasks

The jump in Figure B.2.a) is consistent if the subnet starting at the origin of the jump ($P_1$ in the figure) and finishing at the destination ($P_n$ in the figure) can be delimited (as defined above in the backward jump.) It is assumed that the original net is consistent.

*Proof.*

To proof the above rule the transformation $Trans_1$ in Figure B.2 is used. Making a jump is equivalent to replace the delimited jump subnet in Figure B.2.b) by the transition $T_j$ in Figure B.2.b). Once transition $T_j$ fires the mark is moved from place $P_1$ into place $P_n$ and the jump is accomplished. Therefore, if the final net in Figure B.2.b) is consistent the jump is consistent. Like in the above proof of the jump subnet consistency, the Theorem will be used in the reverse order. Net $N_3$ in the Theorem is the original net (Figure B.2.a)), $N_1$ is the net that results from

$Trans_1$ (Figure B.2.b)) and $N_2$ is the jump subnet. Since subnet $N_2$ replaces the dummy transition $T_j$ in $N_1$ originating $N_3$ the Theorem holds. Then, from the fourth statement in the Theorem and because $N_3$ is by assumption consistent, $N_1$ and $N_2$ are consistent. Hence, the forward jump is consistent.

□

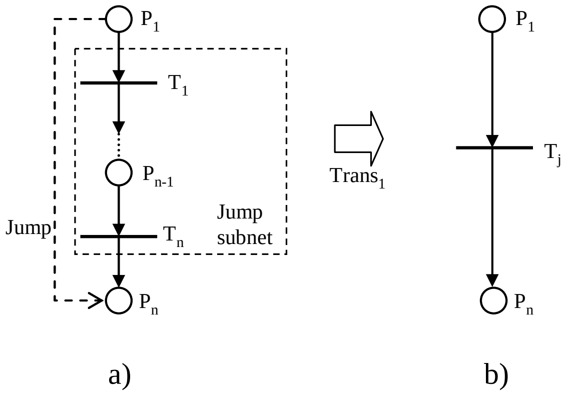


Figure B.2. Equivalent model for the forward jump by skipping tasks

## Forward jump with parallel execution

The forward jump in Figure B.3.a) is consistent if the subnets from $P_1$ to $P_{n-1}$ and from $P_n$ to $P_{m-1}$ can be delimited (as defined above in the backward jump.) The original net is by assumption consistent.

*Proof*

The forward jump with parallel execution of the skipped tasks is equivalent to the transformation of the original model in Figure B.1.a) into the model of Figure B.1.b). The inserted dummy transition $T_{d1}$ simulates the jump operation; once it fires two threads are initiated and task $T_{n+1}$ (the jump destination) can be executed in parallel with the skipped tasks $T_1$ to $T_n$. As in the above proofs, the subnets $T_{n+1}$ to $T_{m-1}$ and $T_1$ to $T_n$ are sequences to keep the figure simple. However, all the statements are valid for any type of net if the mentioned conditions that both subnets can be delimited are met as it will be proofed

To proof the consistency of the jump it is necessary to proof that the model in Figure B.3.b) is consistent when obtained from model in Figure B.3.a). The proof will be obtained by using a series of valid transformations that transform the model in Figure B.3.a) into the model in Figure B.3.b). The first set of transformations removes the two subnets from the original model while the second transformation inserts a parallel thread into the simplified model. Finally, the removed subnets are inserted again into this newly obtained model with the two parallel threads.

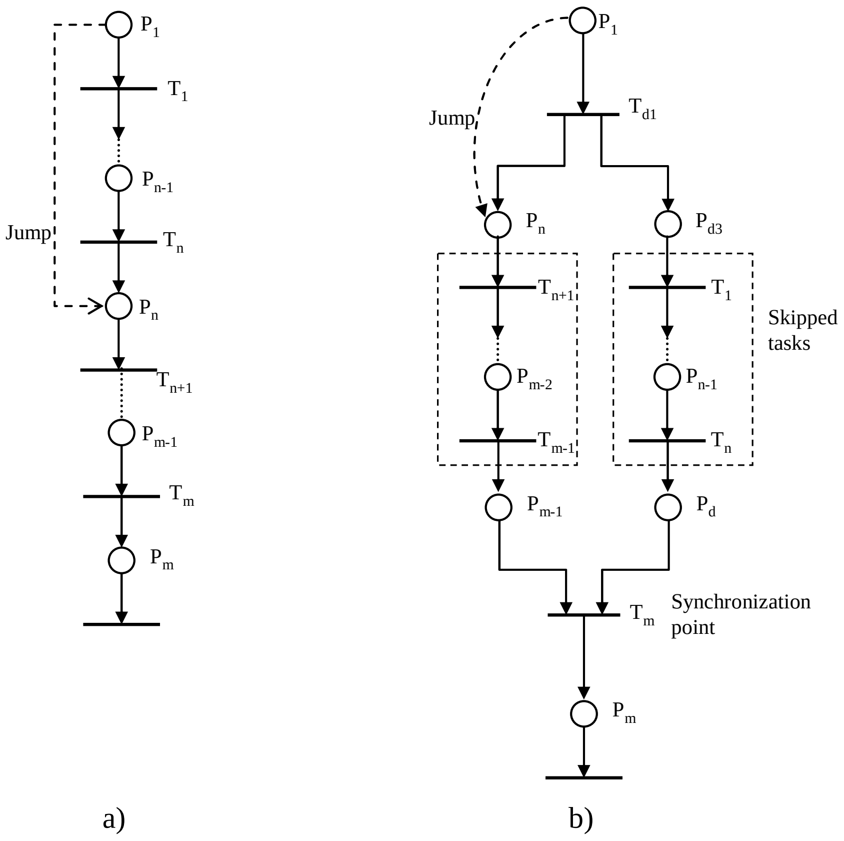


Figure B.3. Equivalent model for the forward jump with parallel execution

Figure B.4. represents the first set of transformations that remove the subnets. In transformation $Trans_1$ the subnet $T_1$ to $T_n$ is removed from the original net while $Trans_2$ removes the subnet $T_{n+1}$ to $T_m$. The last transformation joins two dummy transitions into a single one.

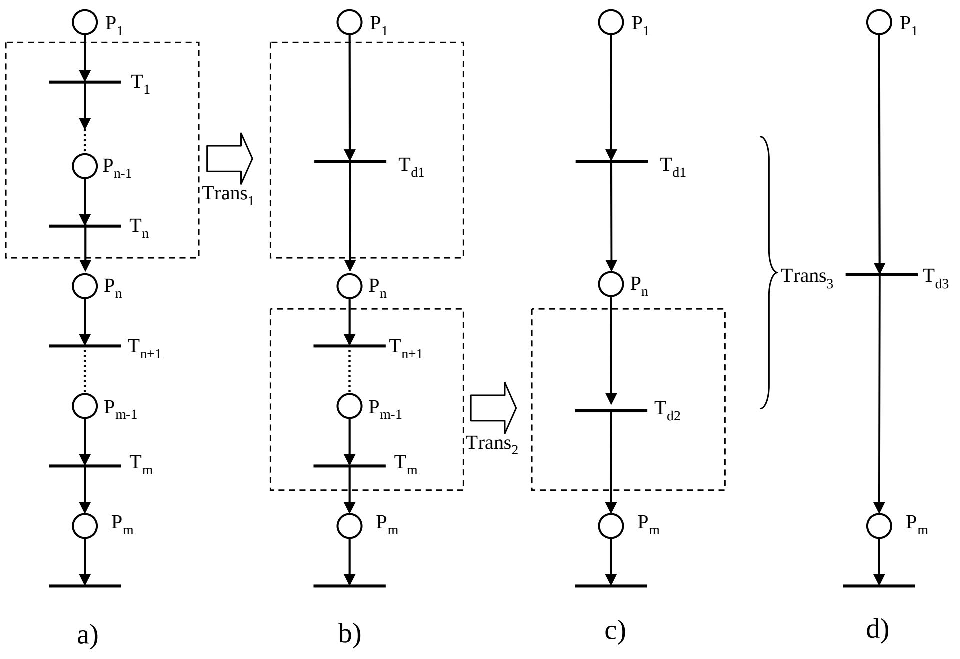


Figure A.4. Transformation series: removing the subnets from the original model

Like it was proofed above for the forward jump with skipped tasks the nets resulting from $Trans_1$ and $Trans_2$ are consistent. Joining the two dummy transitions into a single one is a trivial operation. Therefore, the final net after $Trans_3$ is consistent.

Figure B.5 shows the next transformation that inserts two parallel threads. According to Rule 3 in van der Aalst [van der Aalst, 1997] the transformation shown in the figure transforms a consistent net into another consistent net[17].

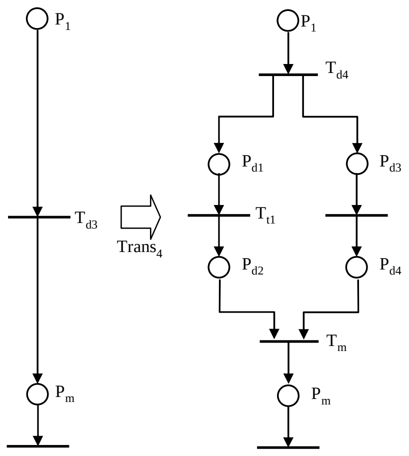


Figure B.5. Transformation series: inserting the parallel threads

Finally, in Figure B.6 the dummy transitions $T_{T1}$ and $T_{T2}$ are replaced by the removed subnets $T_1$ to $T_n$ and $T_{n+1}$ to $T_m$. These are also valid transformations as proofed in the backward jump example since the subnets $T_1$ to $T_n$ and $_{Tn+1}$ to $T_m$ are consistent.

[17] Even though van der Aalst definition of consistency is not as restrictive as the adopted in this thesis it is easy to show that if it preserves live and boundness it also preserves safeness.

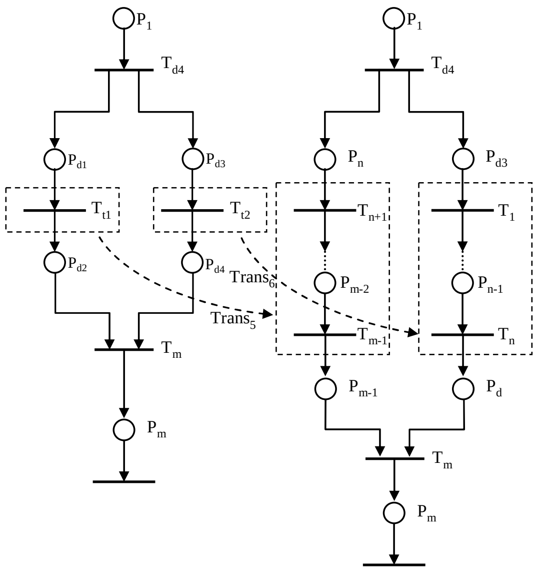


Figure B.5. Transformation series: removing the subnets from the original model

The net resulting from these transformations is the final net of the jump equivalence in Figure B.3.b). Therefore, the final model is consistent and the jump is consistent.

□

# Appendix C – Relationships between diagnosis and recovery

Some relationships can be established empirically between the exception diagnosis dimensions related to *time* and *organizational impact* and the handling strategies *communication type* and *collaboration level*. These relations only intend to be used as suggestions and users may follow the strategy they consider more adequate. Although some field trials should be carried out to validate the usefulness of these relationships, they seem very intuitive and easy to explain. The motivating example is used to explain how they can be useful and to verify their sustainability. These relations may be used by the system to automatically suggest handling strategies to users. For example, if the classifying dimension *time frame to achieve a solution* is set to quick, the system suggests using synchronous communication devices to establish communication.

The relationships are summarised in Table C.1. Since the time associated with the exception is usually an independent factor, the following discussion starts with this dimension. Also, it is important to note that time restrictions have a strong impact on the way people deal with problems. In the diagnosis classes, two dimensions related with time have been defined: *reaction time* and *time frame to achieve solution*. The former is important to specify how the person responsible should be informed about an exception. Then, upon engaging in diagnosis, that person can define the *time frame to achieve solution* in a different way than reaction time; e.g., some monitoring action was immediately implemented but the final solution can be implemented in a more relaxed time frame. Therefore, once the parameter *time frame to achieve solution* is defined, it will have a stronger effect on the decision process than the *reaction time*. The *time* row in Table C.1 reflects this effect and combines these two dimensions into one single variable.

The first row in Table C.1 represents the relations between the *time* dimension on the diagnosis side and the strategies *communication type* and *collaboration level*. If the time dimension is quick, the appropriate strategy to choose is synchronous. Synchronous communication mechanisms allow real-time interactions among the involved actors which is important to reduce the time associated to the exception handling procedure. On the other hand, if time is relaxed or long, the appropriate *communication strategy* is asynchronous because actors may communicate using deferred systems that do not disturb their ongoing activities. It is important to realize that the usage of any *communication type* strategy is not dependent on the *time* value. However, if for any reason that depends on other factors, they decide to use some communication strategy, then they may choose the one referred as the most appropriate in the table.

Table C.1 Relation between diagnosis and handling strategies

| | | Communication type | | Collaboration level | |
|---|---|---|---|---|---|
| | | Synchronous | Asynchronous | Coordinated | Collaborative |
| Time | Quick | Appropriate | | | Appropriate |
| | Relax or long | | Appropriate | Appropriate | |
| Organizational impact | Employee | NA | | NA | |
| | Group or Organization | Choose | | NA | |

The relation between *time* and *collaboration level* strategy also specifies the type of strategy that should be followed. If *time* is quick, users should be involved in a *collaborative strategy* since they may achieve faster response times than they would if they have to coordinate their work with other participants. On the other hand, if *time* is relaxed or long, the *coordinated strategy* is preferred because users are aware of the activities and results implemented by others.

The relations between organizational impact on the diagnosis side and the strategies communication type and collaboration level are represented in the second row of Table C.1. When the organizational impact is employee, it is expected that none of the strategies should be used and hence the not applicable value. If the organizational impact is group or organization, it is expected that some communication strategy is chosen. Finally, when the organizational impact is group or organization, no relation exists with the collaboration level because nothing can be drawn about how actors should collaborate, and hence the value not applicable in the cell.

Even though one example does not validate the proposed relations it can invalidate them if any contradiction is found. It can also be used to discuss their usability on a concrete scenario. The 9/11 motivating example will be used in the remaining of this section to discuss the relations and how they can be useful. Table C.2 and Table C.3 result from applying the proposed classification scheme at the time the second plane hit the tower.

Table C.2. Diagnosis at the time the second plane hit the tower

| Dimension | Value |
|---|---|
| Time | Quick |
| Organizational impact | All organization |

Table C.3. Handling strategies at the time the second plane hit the tower

| Dimension | Value |
|---|---|
| communication type | synchronous |
| collaboration level | collaborative mode involving the whole organization |

The quick value on the *time* dimension forced the synchronous *communication type*, i.e., people tended to share relevant information in real time. The quote from USA Today, “The phone bridges between air traffic facilities have become emergency hotlines and the reports of possible hijackings […] flow at a frenetic pace.”, substantiates this argument. On the other hand, the defined strategy to land all planes adopted a collaborative mode in Memphis Control centre. This strategy was necessary to decrease the coordination overhead: all controllers followed their planes until landing, instead of passing the planes between them. Therefore, the *time* dimension was a critical factor selecting the *communication type* and *collaboration level*, which is according to Table C.1.

Concerning the *organizational impact*, after the Boston controller identified a hijacked situation, he immediately informed the central office in Herndon. The evolution of the situation has to be analysed in more detail. Initially, not many people were involved in the situation and time was relaxed: the plane was not being followed with high priority and no synchronous communication mechanism was being used. This is according to our relation in Table C.1 relating the *organizational impact* with the *communication type*.

Only, when the second plane hit the south tower, and diagnosis dimension *organizational impact* increased to organizational and *time* changed to quick, then the controllers initiated recovery actions and a *collaboration level* had to be established, which is according to our relation.

This final discussion may is important to understand how the relations could be of some use in a situation similar to the example. Using the established relations the air traffic control system could trigger the appropriated handling strategies after diagnosis changes, e.g., after the second plane hit the tower, and the controller in New York changed the diagnosis of reaction time to quick, this information could have been synchronously transmitted to all air traffic control centres because it involved the whole organization. By issuing the event description that generated the new classification everyone would become aware of the situation. Once the communication mechanism is established, the decision to land all planes as fast as possible and to close the USA air space could have been forecast to all controllers instantaneously.

Finally, controllers could have been empowered by the FAA command centre to privilege collaboration among coordination whenever the strategy is safe. As mentioned before, the controllers in Memphis used this strategy to decrease coordination overhead between controllers.

## Appendix D – Citations to our publications

Mourão, H. and P. Antunes (2005) "Supporting Direct User Interventions in Exception Handling in Workflow Management Systems." Sistemas de Informação, 17, pp. 39-51. ISSN: 0872-7031.

(1) Vojevodina, D. (2006) “Modelling of E-Business Process Exception Handling Using Workflow Method”. PhD Thesis. Vilnius Gediminas Technical University. Vilnius.

Mourão, H. and P. Antunes (2005) A Collaborative Framework for Unexpected Exception Handling. Groupware: Design, Implementation, and Use. H. Fuks, S. Lukosch and A. Salgado. Lecture Notes in Computer Science, vol. 3706, pp. 168-183. Heidelberg, Springer-Verlag.

(1) Aveiro, D. and J. Tribolet (2006) Organizational Functions and Enterprise Self-Maintenance: A Framework for Integrating Modelling, Monitoring and Learning. 3rd International CIRP Conference on Digital Enterprise Technology, Portugal.

(2) Aveiro, D. and J. Tribolet (2006) An Ontology for Organizational Functions: The Recursive Self-Maintenance Mechanism of the Enterprise 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06).

(3) Adams, M. (2007) Facilitating Dynamic Flexibility and Exception Handling for Workflows. PhD Thesis. Queensland University of Technology. Faculty of Information Technology. Brisbane, Australia.

Mourão, H. and P. Antunes (2004) Exception Handling through a Workflow. On the Move to Meaningful Internet Systems 2004: Coopis, Doa, and Odbase: OTM Confederated International Conferences, Coopis, Doa, and Odbase. R. Meersman and Z. Tari. Lecture Notes in Computer Science, vol. 3290, pp. 37-54. Heidelberg, Springer-Verlag.

(1) Ardissono, L., R. Furnari, A. Goy, G. Petrone and M. Segnan (2006) Fault Tolerant Web Service Orchestration by Means of Diagnosis. Software Architecture (Ewsa 2006). Lecture Notes in Computer Science, vol. 4344, pp. 2-16, Springer Verlag.

(2) Nepal, S., A. Fekete, P. Greenfield, J. Jang, D. Kuo and T. Shi (2005) A Service-Oriented Workflow Language for Robust Interacting Applications. CoopIS 2005 – International Conference on Cooperative Information Systems Agia Napa, Cyprus.

(3) Rinderle, S., S. Bassil and M. Reichert (2006) A Framework for Semantic Recovery Strategies in Case of Process Activity Failures. 8th International Conference on Enterprise Information Systems, Paphos, Cyprus.

(4) Combi, C., F. Daniel and G. Pozzi (2006) A Portable Approach to Exception Handling in Workflow Management Systems. On the Move to Meaningful Internet Systems 2006: Coopis, DOA, Gada, and ODBASE, OTM Confederated International Conferences, vol. 4275, pp. 201-218. Montpellier, France, Springer.

(5) Adams, M. (2007) Facilitating Dynamic Flexibility and Exception Handling for Workflows. PhD Thesis. Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia.

(6) Jang, J. (2007) Building Reliable and Robust Service-Based Systems for Automated Business Processes. PhD Thesis, School of Information Technologies, University of Sydney, Australia

Mourão, H. and P. Antunes (2003) Supporting Direct User Interventions in Exception Handling in Workflow Management Systems. Workshop de Sistemas de Informação Multimédia e Cooperativos, COOP-MEDIA '03, Porto, Portugal, October 8.

(1) Vojevodina, D. (2005) Exception Handling Automation in E-Business Workflow Processes. The 12th Doctoral Consortium at CAiSE*05, Porto, Portugal.

## Appendix E – Model for the exception handling workflow

```
<workflow>
 <initial-actions>
   <action id="1" name="Instantiate WF">
     <pre-functions>
       <function type="class">
           <arg name="class.name">
               com.opensymphony.workflow.util.Caller
           </arg>
       </function>
       <function type="class">
           <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.NewException
           </arg>
       </function>
     </pre-functions>
     <results>
       <unconditional-result old-status="Finished"
               status="Underway" step="10" owner="${caller}"/>
     </results>
     <post-functions>
       <function type="class">
         <arg name="class.name">
              com.hrm.workflows.StartWfs
         </arg>
       </function>
     </post-functions>
   </action>
 </initial-actions>
 <steps>
   <step id="10" name="Edit exception info">
     <external-permissions>
       <permission name="EditExceptionInfo">
         <restrict-to>
           <conditions type="AND">
             <condition type="beanshell">
               <arg name="script">true</arg>
             </condition>
             <condition type="class">
               <arg name="class.name">
```

```
com.opensymphony.workflow.util.AllowOwnerOnlyCondition
              </arg>
            </condition>
          </conditions>
        </restrict-to>
      </permission>
    </external-permissions>
    <actions>
      <action id="10" name="Start handling">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.opensymphony.workflow.util.AllowOwnerOnlyCondition
              </arg>
            </condition>
          </conditions>
        </restrict-to>
        <pre-functions>
          <function type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
            </arg>
          </function>
          <function type="beanshell">
            <arg name="script">
String resp= (String) transientVars.get("responsible");
System.out.println("action start handling, resp:"+ resp);
            </arg>
          </function>
        </pre-functions>
        <results>
          <unconditional-result old-status="Finished" split="1"/>
        </results>
      </action>
    </actions>
  </step>

  <step id="100" name="Collaboration support">
```

```
    <external-permissions>
      <permission name="Collaborate">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
      </permission>
    </external-permissions>
    <actions>
      <action id="1000" name="Define new responsible">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.opensymphony.workflow.util.AllowOwnerOnlyCondition
              </arg>
            </condition>
          </conditions>
        </restrict-to>
        <pre-functions>
          <function type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
            </arg>
          </function>
          <function type="beanshell">
            <arg name="script">
String resp = (String) transientVars.get("responsible");
System.out.println("action dummy - resp = " + resp);
            </arg>
          </function>
          </pre-functions>
          <results>
            <unconditional-result old-status="Finished"
                    status="Underway"
                    step="110" owner="${responsible}"/>
          </results>
        </action>
        <action id="1010" name="Change affected users">
          <restrict-to>
            <conditions type="AND">
              <condition type="beanshell">
                <arg name="script">true</arg>
              </condition>
              <condition type="class">
                <arg name="class.name">
  com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
                </arg>
              </condition>
            </conditions>
          </restrict-to>
          <pre-functions>
            <function type="class">
              <arg name="class.name">
  com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
              </arg>
            </function>
          </pre-functions>
          <results>
            <unconditional-result old-status="Finished"
                     status="Underway"
                     step="120" owner="${responsible}"/>
          </results>
        </action>
      </actions>
    </step>

    <step id="110" name="Define new responsible">
      <external-permissions>
        <permission name="NewResponsible">
          <restrict-to>
            <conditions type="AND">
              <condition type="beanshell">
            <arg name="script">true</arg>
          </condition>
          <condition type="class">
            <arg name="class.name">
com.opensymphony.workflow.util.AllowOwnerOnlyCondition
            </arg>
          </condition>
        </conditions>
      </restrict-to>
    </permission>
  </external-permissions>
  <actions>
    <action id="1100" name="New responsible defined">
      <restrict-to>
        <conditions type="AND">
          <condition type="beanshell">
            <arg name="script">true</arg>
          </condition>
          <condition type="class">
            <arg name="class.name">
com.opensymphony.workflow.util.AllowOwnerOnlyCondition
            </arg>
          </condition>
        </conditions>
      </restrict-to>
      <pre-functions>
        <function type="class">
          <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
          </arg>
        </function>
      </pre-functions>
      <results>
        <unconditional-result old-status="Finished"
              status="Underway"
              step="100" owner="${responsible}"/>
      </results>
    </action>
    <action id="1110" name="New responsible exit">
      <restrict-to>
        <conditions type="AND">
          <condition type="beanshell">
```

```
            <arg name="script">true</arg>
          </condition>
          <condition type="class">
            <arg name="class.name">
com.opensymphony.workflow.util.AllowOwnerOnlyCondition
            </arg>
          </condition>
        </conditions>
      </restrict-to>
      <pre-functions>
        <function type="class">
          <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
          </arg>
        </function>
      </pre-functions>
      <results>
        <unconditional-result old-status="Finished"
                 status="Underway"
                 step="100" owner="${responsible}"/>
      </results>
    </action>
  </actions>
</step>

<step id="120" name="Change affected users">
  <external-permissions>
    <permission name="ChangeAffectedUsers">
      <restrict-to>
        <conditions type="AND">
          <condition type="beanshell">
            <arg name="script">true</arg>
          </condition>
          <condition type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
            </arg>
          </condition>
        </conditions>
      </restrict-to>
    </permission>
  </external-permissions>
```

```
    <actions>
      <action id="1200" name="Finish change affected users">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
        <pre-functions>
        <function type="class">
          <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
            </arg>
          </function>
        </pre-functions>
        <results>
          <unconditional-result old-status="Finished"
                  status="Underway"
                  step="100" owner="${responsible}"/>
        </results>
      </action>
    </actions>
  </step>

  <step id="200" name="Exception description">
    <external-permissions>
      <permission name="ChangeInstances">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
      </permission>
      <permission name="ExceptionClassification">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
      </permission>
    </external-permissions>
    <actions>
      <action id="2000" name="Change association of instances">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
        <pre-functions>
          <function type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
            </arg>
          </function>
        </pre-functions>
        <results>
          <unconditional-result old-status="Finished"
```

```
                 status="Underway"
                 step="210" owner="${responsible}"/>
        </results>
      </action>
      <action id="2010" name="Edit exception classification">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
        <pre-functions>
          <function type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
            </arg>
          </function>
        </pre-functions>
        <results>
          <unconditional-result old-status="Finished"
                         status="Underway"
                         step="220" owner="${responsible}"/>
        </results>
      </action>
    </actions>
  </step>

  <step id="210" name="Change association of instances">
    <external-permissions>
      <permission name="ChangeInstances">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
```

```
                <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
                </arg>
            </conditions>
          </restrict-to>
        </permission>
      </external-permissions>
      <actions>
        <action id="2100" name="Finish association of instances">
          <restrict-to>
            <conditions type="AND">
              <condition type="beanshell">
                <arg name="script">true</arg>
              </condition>
              <condition type="class">
                <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
                </arg>
              </condition>
            </conditions>
          </restrict-to>
          <pre-functions>
            <function type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
              </arg>
            </function>
          </pre-functions>
          <results>
            <unconditional-result old-status="Finished"
                        status="Underway"
                        step="200" owner="${responsible}"/>
          </results>
        </action>
      </actions>
    </step>

    <step id="220" name="Exception classification">
      <external-permissions>
        <permission name="ExceptionClassification">
          <restrict-to>
            <conditions type="AND">
          <condition type="beanshell">
            <arg name="script">true</arg>
          </condition>
          <condition type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
            </arg>
          </condition>
        </conditions>
      </restrict-to>
    </permission>
  </external-permissions>
  <actions>
    <action id="2200" name="Finish exception classification">
      <restrict-to>
        <conditions type="AND">
          <condition type="beanshell">
            <arg name="script">true</arg>
          </condition>
          <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
            </arg>
          </condition>
        </conditions>
      </restrict-to>
      <pre-functions>
        <function type="class">
          <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
          </arg>
        </function>
      </pre-functions>
      <results>
        <unconditional-result old-status="Finished"
                status="Underway"
                step="200" owner="${responsible}"/>
      </results>
    </action>
  </actions>
</step>
```

```
    <step id="300" name="Recovery actions">
      <actions>
        <action id="3000" name="Execute a recovery action">
          <restrict-to>
            <conditions type="AND">
              <condition type="beanshell">
                <arg name="script">true</arg>
              </condition>
              <condition type="class">
                <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
                </arg>
              </condition>
            </conditions>
          </restrict-to>
          <pre-functions>
            <function type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
              </arg>
            </function>
          </pre-functions>
          <results>
            <unconditional-result old-status="Finished"
status="Underway"
                  step="300" owner="${responsible}"/>
          </results>
        </action>
      </actions>
    </step>

    <step id="400" name="Monitoring actions">
      <actions>
        <action id="4000" name="Insert monitoring action">
          <restrict-to>
            <conditions type="AND">
              <condition type="beanshell">
                <arg name="script">true</arg>
              </condition>
              <condition type="class">
                <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
```

```
            </arg>
          </condition>
        </conditions>
      </restrict-to>
      <pre-functions>
        <function type="class">
          <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
          </arg>
        </function>
      </pre-functions>
      <results>
        <unconditional-result old-status="Finished"
status="Underway"
                step="400" owner="${responsible}"/>
      </results>
    </action>
   </actions>
  </step>

  <step id="500" name="External info">
   <external-permissions>
     <permission name="GenerateApplicationData">
      <restrict-to>
        <conditions type="AND">
          <condition type="beanshell">
            <arg name="script">true</arg>
          </condition>
          <condition type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
            </arg>
          </condition>
        </conditions>
      </restrict-to>
     </permission>
     <permission name="GenerateExternalDecision">
      <restrict-to>
        <conditions type="AND">
          <condition type="beanshell">
            <arg name="script">true</arg>
          </condition>
```

```
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
      </permission>
    </external-permissions>
    <actions>
      <action id="5000" name="Insert external information">
        <restrict-to>
          <conditions type="AND">
            <condition type="beanshell">
              <arg name="script">true</arg>
            </condition>
            <condition type="class">
              <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.OSUserAffected
              </arg>
            </condition>
          </conditions>
        </restrict-to>
        <pre-functions>
          <function type="class">
            <arg name="class.name">
com.hrm.apss.dominial.opensymphony.exceptions.ResponsibleOwner
            </arg>
          </function>
        </pre-functions>
        <results>
          <unconditional-result old-status="Finished"
                  status="Underway"
                  step="500" owner="${responsible}"/>
        </results>
      </action>
    </actions>
  </step>

<!--END STATE -->
  <step id="10000" name="WF Finished">
    <actions>
```

```
      <action id="10000" name="End" auto="true" finish="true">
        <results>
          <unconditional-result old-status="Finished"
status="Finished"
                 step="200"/>
        </results>
      </action>
    </actions>
  </step>

 </steps>
 <splits>
   <split id="1">
     <unconditional-result old-status="Finished"
             status="Underway"
             owner="${responsible}" step="100"/>
     <unconditional-result old-status="Finished"
             status="Underway"
             owner="${responsible}" step="200"/>
     <unconditional-result old-status="Finished"
             status="Underway"
             owner="${responsible}" step="300"/>
     <unconditional-result old-status="Finished"
             status="Underway"
             owner="${responsible}" step="400"/>
     <unconditional-result old-status="Finished"
             status="Underway"
             owner="${responsible}" step="500"/>
   /split>
 </splits>
 <joins>
   <join id="1">
     <conditions type="AND">
       <condition type="beanshell">
         <arg name="script"><![CDATA[
                "Finished".equals(jn.getStep(200).getStatus()) &&
                "Finished".equals(jn.getStep(300).getStatus())]]>
         </arg>
       </condition>
     </conditions>
     <unconditional-result old-status="Finished"
status="Underway"
               step="1000"/>
    </join>
  </joins>
</workflow>
```