WIRES: A Methodology for Developing Workflow Applications

Fabio Casati\textsuperscript{a}, Maria Grazia Fugini\textsuperscript{b}, Isabelle Mirbel\textsuperscript{c} and Barbara Pernici\textsuperscript{b}

\textsuperscript{a}Hewlett-Packard Laboratories, Palo Alto, California, USA; \textsuperscript{b}Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy; \textsuperscript{c}Laboratoire I3S, Les algorithmes, Universite de Nice, Sophia Antipolis, France

Workflow management systems are becoming a relevant support for a large class of business applications, and many workflow models as well as commercial products are currently available. While the large availability of tools facilitates the development and the fulfilment of customer requirements, workflow application development still requires methodological guidelines that drive the developers in the complex task of rapidly producing effective applications. In fact, it is necessary to identify and model the business processes, to design the interfaces towards existing cooperating systems, and to manage implementation aspects in an integrated way. This paper presents the WIRES methodology for developing workflow applications under a uniform modelling paradigm – UML modelling tools with some extensions – that covers all the life cycle of these applications: from conceptual analysis to implementation. High-level analysis is performed under different perspectives, including a business and an organisational perspective. Distribution, interoperability and cooperation with external information systems are considered in this early stage. A set of 'workflowability' criteria is provided in order to identify which candidate processes are suited to be implemented as workflows. Non-functional requirements receive particular emphasis in that they are among the most important criteria for deciding whether workflow technology can be actually useful for implementing the business process at hand. The design phase tackles aspects of concurrency and cooperation, distributed transactions and exception handling. Reuse of component workflows, available in a repository as workflow fragments, is a distinguishing feature of the method. Implementation aspects are presented in terms of rules that guide in the selection of a commercial workflow management system suitable for supporting the designed processes, coupled with guidelines for mapping the designed workflows onto the model offered by the selected system.

Keywords: Extensions to UML; Methodology; Reusable patterns; Workflow; Workflows for e-services

1. Introduction

Workflow management systems (WFMSs) are software systems for the definition, execution and management of business processes [1]. WFMSs are currently widely used to support the implementation of a large range of applications requiring the execution of distributed processes, possibly spanning across several systems and several companies. Traditionally, workflow technology has been applied to support the flow of documents in administrative operations. More recently, this technology has been successfully extended to automate production processes, involving the execution of complex transactions on top of heterogeneous, autonomous and distributed systems, as well as the composition of e-services.

Despite the wide diffusion of WFMSs, methodologies covering all the phases of development of workflow applications are still missing. Existing proposals concentrate on specific fields of workflow development, such as specification languages for structured workflows [2] (i.e., workflows with restrictions on constructs), or paradigms for defining and managing concurrency and coordination among workflows (e.g., in [3]). Other approaches are related to the integration of WFMSs with ERP products, such as SAP [4]), or are closely related to commercial workflow management systems.
The WIRES methodology presented in this paper allows designers first to consider, in the analysis phase, the conceptual aspects of the application, such as the business goals and the identification of which processes should be implemented as workflows. In this phase, the interactions between the business process and the existing information systems are also modelled. In the design phase the developer can design the workflow schema and its transactions. Finally, the methodology guides the developer in mapping the designed schema into the language supported by the selected WfMS, leading to a complete development of an operational system.

WIRES expands the methodological steps Baresi et al. [5] proposed in WIDE. It follows the three basic steps initially proposed (analysis, design, and mapping to implementation), but adopts a single notation, namely an extension of UML (Unified Modeling Language) [6]. The use of UML makes the methodology compliant with object-oriented notations, and can be employed together with object-oriented software development methodologies [7], also supporting software development by components reuse. Therefore, WIRES has been conceived to support workflow development based on the reuse of components, which is a widely accepted paradigm in software engineering (see [8,9] for a survey).

We will adopt some of the recently proposed extensions to UML for business process modelling (e.g., [4]) in order to meet the WIRES goals listed above and will propose some extensions of our own. We will show how issues, such as cooperating and distributed workflows, and interaction with external information systems, can be managed using UML; we will also show some additional notations needed to support development of exception handling and transaction management.

In the literature, one of the first approaches to systematic workflow development is ActionWorkflow [10], where the design is based on the iteration of proposal, agreement, performance and satisfaction loops. Under this approach, it may be difficult to systematically derive the activity flow, in particular when ActionWorkflow loops must be mapped onto a conventional workflow model, based on a procedural description of the activities.

WfMS vendors have recently started to provide methodological support to the design of workflows to be implemented in their systems, such as in Hewlett-Packard [11]. These approaches are mainly focused on the use of a particular model and on sample usages of the different constructs to design workflow applications.

Many approaches to workflow design have a strong emphasis on providing a procedural and possibly graphic representation of the activity flow [12,13]. These approaches are suitable for modelling the normal execution of a process, defined by the sequencing of activities, but are not suited for modelling exceptional situations, which are often asynchronous with the progression of the workflow.

Other methodologies (e.g., [14]) are limited to the specification of workflows at a conceptual level. They involve the use of several informal or semi-formal tools to model different aspects of the workflow, such as the information aspect, the process aspect and the organisational aspect. It is widely recognised that a fragmented conceptual view is useful for the purpose of systems analysis and documentation, but no guidelines are provided for the construction of the workflow [12].

In Bajaj and Ram [15], the authors merge workflow conceptual views, useful for systems analysis and documentation, with design aspects, to automate the design and management of workflows. The proposed SEAM methodology gives a conceptual workflow model in terms of set theory and incorporates temporal aspects. A prototype application shows how SEAM workflow schemas can be implemented on a relational database management system. Organisational issues are discussed in terms of the benefits of SEAM for the developers and the organisation. However, a true organisational model is not tackled. Moreover, potential advantages in using the SEAM methodology to build real workflow systems are discussed, but are not introduced as an explicit step in the methodology.

In WIRES, we define a complete methodology, which includes a comprehensive set of steps ranging from business analysis to implementation mapping, with focus on distributed transactions, anomalies and development through reuse. The proposed methodology is suited for supporting administrative, human-oriented processes as well as production workflows used to implement e-services. The case study presented in this paper is related to an e-service application.

The goals of the WIRES methodology can be summarised as follows:

1. To propose a sequence of steps that allow developers to consider business processes and existing information systems and produce an integrated workflow schema manageable by a commercial WfMS, endowed with transactions and exception-handling mechanisms.
2. To provide reusable workflow frames, or components, analogous to software components [8,16,17], and a method to compose these frames into a new
workflow starting from ‘component’ workflow schemas previously developed in similar domains.

3. To guide the design of an integrated next-generation workflow development environment and related tools that can be tested in real-world application scenarios.

The paper is organised as follows. We start with an overall presentation of the WIRES methodology in Section 2 and illustrate its phases: workflow analysis, design and mapping to implementation. We show how different levels of detail are provided in the different phases, giving rules to progressively enrich the UML analysis schema with design elements. An example used along all phases is introduced.

Section 3 presents the analysis phase, which, starting from business goals and resource descriptions, originates UML use case diagrams, enriched with an organisational model. We also present ‘workflowability’ criteria to decide whether the business process, or portions thereof, can be implemented as workflows and how the interactions with the enterprise information system should occur. Section 4 describes the design phase, which considers those portions of the business process that have been selected as ‘workflowable’ in the analysis. Design is centred around UML activity diagrams, progressively refined with task specification, exception specification, transaction management aspects and interactions with external applications. In this section we also discuss design based on the reuse of workflow components as well as aspects related to distributed workflows.

We then turn to technology-related aspects. Section 5 presents the basic criteria to be followed when selecting a commercial WfMS compliant with the workflow produced in the design phase, and the rules that allow the designer to map the designed schemas into an operational workflow application. Finally, in Section 6, concluding remarks and future developments are given.

2. Purposes and Phases of the Methodology

2.1. Addressed Problems and Featuring Issues of WIRES

Workflow development has to integrate aspects that range from high-level, business and end-user-oriented aspects to implementation issues, such as the functionalities of commercial WfMSs, the physical distribution of work and resources, system interoperability and compatibility with external applications. In the following we discuss these aspects and their implications for a systematic development of workflow applications:

- **Bridging the gap between conceptual and implementation levels.** Methods and tools for developing conceptual-level specifications, as well as methods for mapping conceptual workflow schemas into the languages supported by commercial WfMSs are still lacking, or lack integration from the business process modelling to workflow modelling and implementation.

- **Uniform modelling paradigm.** A uniform model in all the phases of workflow development facilitates the mapping among phases and the iteration development.

- **Reuse of components.** WIRES applies reuse of workflow components. **Workflow components** are workflow fragments produced in previous developments, which have been engineered and made available in structured repositories. Workflow components are identified in the analysis phase, instantiated and composed during the design phase, and made executable in the implementation phase.

In the WIRES methodology we adopt an approach based on a library or repository of workflow components, called patterns, that we have developed in previous work [18,19]. The library contains patterns that model fine-grained components (e.g., exceptions) and coarse-grained components such as a whole subprocess, with associated actors, data, and exceptions. In Section 4, we mention the WIRES mechanisms to reuse patterns and to tailor them in a workflow application. The reader is instead referred to Casati et al. [19] for details on pattern tailoring and on problems such as the choice of the granularity of components.

- **Non functional requirements.** Non-functional features of business processes, such as the repeatability of the process, the number of persons needed to execute a process, or the need of managing exceptions, play an important role in identifying candidate workflows, that is, the business processes that will be implemented with a WfMS. WIRES considers non-functional features since the early stages of development.

After these general considerations, we now describe the phases of WIRES.
2.2. WIRES Phases

While keeping a quite traditional sequence of development phases (analysis, design, and mapping to implementation), WIRES is based on some peculiar aspects deriving from the following considerations about workflow development presented in the previous paragraphs. The overall structure of the WIRES development methodology for workflow applications is shown in Fig. 1.

- The analysis phase aims at defining the business processes that can be implemented as workflows, the system boundary, and the interactions with other workflows and external information systems (legacy applications). The analysis also outlines the organisation of processes into sub-processes and identifies the actors, the distribution and interoperability issues, and the possibility of reusing workflow components.

UML use cases and sequence diagrams (scenarios) are used to model the main processes and sub-processes. The extension to UML proposed by WIRES regards the business context. Since the business perspective is somehow lacking in UML, we require a list of business goals to be associated with business processes in the form of a use case/goal matrix, showing the correspondence between goals and use cases. The business context is completed by specifying how the organisation units are structured and how they cooperate, using the UML package notation. Packages allow us to analyse cooperation among organisation units, and to derive distribution requirements.

External or existing information systems (ISs) are represented as actors, and their interactions with the business process are modelled in use cases and in sequence diagrams.

Distribution and interoperability aspects are given in the form of sequence diagrams and/or collaboration diagrams. The sequence diagram explains the exchange of messages between the distributed parts of a sub-process. UML collaboration diagrams are used to model the objects and links that are meaningful within a distributed interaction. For non-functional aspects, 'workflowability' criteria are provided.

An analysis of reuse potential of available components for the application to be developed is also performed in this phase.

- The design phase aims at producing a (set of) workflow schema(s), where the flow of activities and the exceptional situations are represented. The interactions of the workflow with external applications and information systems are detailed. The design phase is based on extended UML activity diagrams. We build on recent extensions to these diagrams [4,6], with the aim of representing events (temporal, external and data events), multi-tasks, transactional aspects and temporal constraints. Reuse of patterns is detailed in this phase, with the actual selection of the reuse components (indicated in the analysis) from the library, and the application of reuse primitives that allow the designer to insert instantiated and customised patterns in the workflow schemas.

- The mapping phase guides the implementation choices regarding the selection of a WfMS. The phase gives a set of criteria for mapping the design elements of the workflow schema into the features of commercial WfMSs and for obtaining an executable workflow given in the target WfMSs.

The choice of a standard set of phases (analysis, design, and mapping to implementation) implies that WIRES can be used stand-alone or it can be coupled with standard software and IS development methods.
2.3. A Reference Example

An example is now introduced to illustrate WIRES throughout the paper. The example refers to the ‘DVD Smart Centre’ Company, partly taken from a real test case. The company business consists in allowing customers to buy DVDs by accessing distributed retail sites. Suppose the retail sites, of different size and relevance in terms of available products, exist and are networked to manage orders in a distributed way. This means that an order whose items are not available at a given location can be processed remotely by searching the DVDs in other shops and delivering the DVDs via a shipping service to the customer. Therefore, we assume a set of information systems (External IS in Fig. 1) already exist to process orders, payments and stocks, with databases regarding both customers and goods.

Assume that, to increase revenues and reduce costs, the ‘DVD Smart Centre’ company wants to enable Web orders and automate the order fulfillment process. Customers will be able to browse the online DVD catalogue and submit orders by specifying the DVDs of interest, packaging requirements, destination addresses, payment modalities and other details. First, we give a textual explanation of the process, and then we summarise its main characteristics.

The process starts when an order is submitted. The customer’s order data are extracted from the Web form. Then information about the customer is collected. If the customer data (e.g., name, address and payment information) are already stored in the customers’ database, only the authentication data are required (userId and password), and the customer profile record is read with preferences and information about previous interactions. If the authentication fails, an error is notified to the responsible. After authentication, the order is validated (all the required fields must be present). If the check fails, the process terminates. Otherwise, the customer profile is updated, or created if the customer is new; an email confirmation of profile creation is sent back to the new customers. If an error on the customer’s profiles database occurs, an exception is raised and notified to the responsible clerk.

If everything is OK, the order can be processed, i.e., picked and checked against the stocks. An exception is raised if the ordered item is out of stock and an external reordering process is started. If a DVD is not available at the inspected site, the order is processed in a distributed way, by automatically forwarding order data to other retail sites. When the ordered DVD is located, an email confirmation and a payment request are sent to the customer. When the payment is confirmed, the delivery is started (or waited for, if the item is out of stock) and the order is shipped using an external transportation company (here the interaction with an outsourced service occurs).

Figure 2 summarises the process elements of interest.

3. Workflow Analysis

The first step of WIRES is workflow analysis, addressing the question ‘Which business processes can or should be supported by workflow technology?’. We assume that the business processes examined in the analysis phase have been derived by process analysts, possibly by using the methods proposed in the literature (based on interviews, surveys, team discussions, and so on – see [20] for a discussion). Since an automatic transfer of business process models into workflow models proves impractical [20], an intermediate phase is needed for analysing the organisational structure and the existing technical structure of the company, i.e., existing (legacy) systems and system distribution issues.

During workflow analysis, requirements are refined and structured, using the language of the developers and treating issues regarding the system activities and roles, the anomalous cases, the distribution and cooperation requirements, and the relationships with other systems. Workflow analysis has both some ‘generic-type’ aims, common to all complex software systems developments, and some ‘workflow-specific’ aims that are discussed briefly in the next paragraph.

3.1. Analysis Aims

A first aim consists in identifying the boundaries of the workflow application, i.e., the business requirements that motivate it, and its relationships with other system areas or real-world portions that are already covered by existing systems or that do not need to be automated. Fulfilment of this aim needs to identify processes and roles that are suited to be implemented through workflow technology, due to their functional and non-functional characteristics.

While for generic software systems functional specifications are generally sufficient, at least in the analysis phase, because the system behaviour is the central matter, a typical issue of workflow development is that the analysis strongly requires non functional requirements, because these are often a key point to decide whether a business process should become a workflow application instead of a conventional information system application. Modelling tools are needed to identify and specify the two types of requirements and to classify them in order to provide enough information to
the design and mapping phases and minimise further cycles back.

In WIRES, functional requirements are modelled through \textit{UML use cases} with scenarios (\textit{sequence diagrams}). Different levels of detail for use cases and sequence diagrams can be given, for example only for the main sub-processes. WIRES keeps the analysis details at a high level, sufficient to understand the main steps of the business process, its actors, and the distribution and interoperability aspects.
Non-functional requirements are taken into account by compiling UML textual annotations and by giving a list of criteria suitable for analysing the ‘workflowability’ of a process.

A second aim of the analysis is to focus the business goals from the initial requirements, taking into account the context of the business process, i.e., the application domain. UML is extended to show the correspondence between business processes and goals. Business goals support the identification of business processes to be automated through workflow technology.

The main output of the analysis phase is a list of candidate workflows (see Fig. 1), that is, the (parts of) business processes that, according to ‘workflowability’ criteria, are suitable to be implemented as workflows. Besides, the correspondence between business goals and the identified candidate workflows is shown in a use case/goal matrix. The main actors involved in candidate workflows are also identified.

A third aspect in the analysis regards the possibility to get a first, general, idea of how the workflow is organised into sub-processes to be implemented as tasks of the workflow, or even as separate workflows, due, for example, to geographical distribution. Distribution aspects (regarding those sub-processes that are executed or interact in a distributed way) are represented giving scenarios (sequence diagrams) for the portions that cooperate and identifying the messages (in a time sequence) that model the cooperation. In our reference example, Process Order is a distributed process since it is executed over different retail sites, depending on the product availability or on the proximity of the site to the customer. The choice of the retail site can be made dynamically by the system. Distribution is modelled through a sequence diagram where the distributed Process Order is started via a message broadcast protocol over the network.

The Delivery-Ship Order sub-process is instead an example of process executed cooperatively between the internal Delivery service and an outsourced service. Cooperation is also modelled as a distributed process through a sequence diagram; here, synchronisation messages are used to ensure that Delivery and Shipping are coherently executed.

A fourth aim consists in modelling business processes in a way that is significant also for the customer, based for instance on graphical notations, and to show clearly the users involved in the business processes and in the workflows activities. Considering the modelling aspect, we observe that nowadays the construction of complex systems uses object-oriented analysis and design, while workflow design is still performed separately. Therefore, the trend is to adopt object-oriented techniques and tools for the whole system for describing the structure and the business processes of an enterprise, also for re-engineering and reuse purposes [21–23].

In the analysis, classes represent an abstraction of one of several classes and/or subsystems in the workflow development: business processes, workflow data, and interaction between the system and its actors (users and external systems).

However, in standard UML, analysis focuses on functional requirements and postpones non-functional requirements to design and implementation activities. Since we need to gain information about aspects such as the amount of work involved in a process, the number of people, the frequency of execution and so on, we use UML annotations to represent non-functional requirements starting from the analysis. For instance, in our example ‘DVD Smart Company’, sample non-functional requirements are the following:

1. When the system actors (e.g., employees) ask to display an order note, it should not take more than 0.3 seconds to visualise the order.
2. Orders should be confirmed by the customer using an authentication protocol (e.g., providing an authentication certificate).

As a uniform and object-oriented modelling method, UML can support development by reuse of workflow components. Although reuse of suitable workflow components (e.g., workflow frames, basic exception handlers, reusable roles) is actually performed in the design phase, the analysis aims at early identification of repeated, common situations in workflows [19].

In our example, a Validate Order procedure and a database error handler are, even though at different levels of granularity, common elements that are likely to belong to a library of reusable patterns. The aim of the analysis is to identify such common frames and to identify possible patterns, considering that the analyst has a good knowledge of the development environment he is coping with and therefore of the contents (at least semantically) of the reuse library.

### 3.2. Perspectives

The analysis phase receives as an input the description of business processes according to three perspectives:

- **the functional perspective**, a high-level description of process activities and involved information objects;
- **the organisational perspective**, describing actors and roles involved in process execution;
- **the business perspective**, describing business goals related to the process.
3.2.1. Functional Perspective

The functional perspective focuses on the operational structure of the business process. The core description is a high-level representation of the business process through use case diagrams, with examples of use of the business process (scenarios) through sequence diagrams. Use case and sequence diagrams are usually employed in UML as basic notations for business process requirements [24].

The use case diagram representing the whole business process identifies: (i) the boundary of the system, and (ii) the actors involved in the business process and their interactions with the system. As we are dealing with WF analysis, it is important to distinguish the applications and organisational roles interacting with the WF process from the external entities the process is related to. Therefore, we define two stereotypes associated with the actors: internal and external. Internal actors represent organisational roles or applications that interact directly with the process within the company. External actors are entities (users or systems) the process is dependent on, for example through sharing of a database. These two kinds of actor stereotypes are graphically represented through different icons.

The use case diagram comprises also the representation of the main sub-processes of the business process and their interactions with actors, to express a usage pattern, an activation, or a communication between an actor and the use case.

Each sub-process in the use case diagram can be further detailed through a use case diagram of a first-level refinement, considering the initial representation a zero-level diagram. In Fig. 3, the use case of the ‘DVD Smart Company’ is given. It is a zero-level representation. External actors are represented as black filled actors icons.

We have outlined the main sub-processes (ovals) of the overall system. The use case also visualises the external systems (Web and email) through which the DVD business process dialogues with the customer (catalogue browsing, ordering, and payment submission operations). We have simplified by unifying the Customer&Payment information systems into a unique system that manages customers’ profiles and invoices.

We also model as actors the information systems which, with their Customers, Payment and Order databases, interact with the business process mainly to exchange data stored in these databases. The Delivery-Ship Order process is split between an internal delivery service and an external outsourced Shipping system that cooperate. This is shown here by representing generically an external Shipping information system; details are given in a specific sequence diagram.

The internal actors are the Responsible, who is in charge of managing the validated Order, and the Customer Relationship Office-CROOffice, which acts as a call centre for the customers. At the zero level, we do not represent notification of errors and anomalies. These anomalies and notifications are represented in other refinement levels where the sub-processes are detailed.

An example of a first-level refinement use case is given in Fig. 4 regarding the Get Customer’s Order sub-process. This sub-process collects data about the customer (stored in the external Customer&Payment IS). If the data (e.g., name, address and payment information) are already stored, then only the authentication data are required (userId and password) and then the customer profile record is read, with preferences and information about previous interactions. If the authentication fails, an error is notified to the responsible. A stereotype is used for this kind of message, for instance, <<notify>>. Next, the order is checked in order to verify that no required information is missing and that the credit card number/expiration date is valid. If the order is not valid the process is terminated, otherwise the customer’s profile is registered and an email of confirmation that he/she has been added to the customer base is sent. We have represented the situation in which a new customer is registered as an exceptional one (stereotype

![Fig. 3. Use Case of the ‘DVD Smart Company’ (zero-level).](image-url)
Database errors, e.g. due to false data, are notified to the responsible (the notification actions are depicted with stereotypes).

Also distribution is outlined in the analysis. For example, the sub-process Process Order of Fig. 3 is detailed in a first-level refinement sequence diagram where broadcast messages are sent over the retail sites network to search for the desired DVD. Criteria for searching on the net are the non-availability of the needed DVD or optimisation criteria, such as the closeness of the retail site to the customer. Fig. 5 shows a simplified sequence diagram outlining how Process Order interacts with networked retail sites and with the Web system.

Now, a set of scenarios is associated with each sub-process to simulate the use case diagram. Scenarios allow the representation of the control flow aspects that could not be derived from the use case diagram. Scenarios, represented by UML sequence diagrams, describe how the involved entities (actors and sub-processes) cooperate in a specific sample situation.

In WIRES, scenarios describe both the ‘normal’ behaviour and the ‘exceptional’ behaviour of a given process or sub-process. Exceptional behaviour is given by the occurrence of anomalous or unusual events, such as errors or interruptions to the execution flow. If several exceptional behaviours are identified in the same sub-process, different scenarios must be provided, each describing an exceptional behaviour of the sub-process itself. Anomalous situations can be treated by reusing predefined exceptional cases. Exceptional scenarios should be defined to show how the system behaves in correspondence of decision points: possible alternative behaviour(s), which are not considered in the ‘normal’ scenario, must be supplied to explain how the process manages these exceptions.

In Fig. 6, the Normal Scenario of the Get Customer Data sub-process is depicted. In Fig. 7, an example of exceptional scenario regarding the same sub-process is depicted, related to handling a new customer and to possible authentication errors. In this example, the situation of the new customer and of handling his/her security data is treated as a deviation from the normal flow due to a modelling choice, since WIRES provides exceptions as a mechanism to design such deviations.
Relevant business data, such as information about customers, orders, invoices, payments, are represented through UML classes.

3.2.2. Organisational Perspective

According to this perspective, UML is used to model aspects related to actors and roles as well as to organisational units involved in a business process. Agents are physical entities that play one or more roles, while roles express collections of responsibilities. In UML, the concept of actor is used to model the different roles involved in a business process. Actors are classes with a specific stereotype (icon); i.e., business process roles are specified as entities with a state and a behaviour. Actor classes can be related to each other through relationships modelling the different relationships holding among business process roles (e.g., hierarchical relationships). The different organisational units involved in the business process are modelled using the UML concept of package. The overall organisation is represented through a set of cooperating packages, each comprising a set of actors among the identified ones.

In Fig. 8, actors and organisational units for our example are shown, describing the organisational perspective of the ‘DVD Smart Company Order Processing’. The outsourced service of shipping is represented as an external organisational unit interacting with the internal units via an interface.

3.2.3. Business Perspective

Goals and sub-goals have to be provided through a hierarchy of classes using UML. The most generic classes represent the high-level goals, while the most specific classes represent the specific sub-goals. Such a diagram may also be completed by constraints over the generalisation hierarchy, as provided by the UML notation. Additional information about the relationships among the different subgoals may be given through these constraints. For instance, it can be specified that one goal has been subdivided into different subgoals, and that these goals fully cover the most general one. Such a constraint is specified using the complete constraint. In our example, Improve technology level of relationships with customer, Minimise waiting times, Ensure order fulfilment over distributed system and Ensure security and privacy are the four sub-goals fully covering Order processing goals (See Fig. 9).

In WIRES, a sub-goal should be associated with a sub-process. In particular, sub-goals should be related to the activities appearing in the use case diagram(s). A use case/goal matrix is used to show the correspondences between goals and use cases. It helps to find a suitable representation of the sub-processes. Goals will be considered in the design phase, focusing on how they can be achieved by suitably structuring the flow of activities or by designing specific exceptions.

A use case/goal matrix for our reference example is depicted in Fig. 10, describing the overall business perspective of the ‘DVD Smart Company Order Processing’.
3.3. Selection of Candidate Workflows and Identification of Reusable Components

The goal of this step is to understand whether the business process can be supported by workflow technology. Basically, WfMS support is useful if the benefits, typically measured in terms of reduced operating costs (e.g., due to reduced use of human resources) and improved process execution quality and speed are superior with respect to the costs necessary to implement the workflow-based solution. In particular, workflow technology is applicable if the processes are repeatable, since the effort required to define, test and deploy the process can be exploited by many executions. In addition, in some cases, manual scheduling and assignment can be difficult and subject to constraints that must be enforced. Processes with these characteristics are typically good candidates for workflow-based automation. WfMSs are also exploited to support processes that require monitoring and exception detection (for example, to manage deadline expiration by notifying the appropriate users and possibly performing automated reactions).

Other requirements and opportunities that push companies to adopt workflow technology are the need for tracking process executions (for instance, to be able...
to promptly give information to customers inquiring about the status of their orders) and for analysing completed execution, in order to optimise the process. Fig. 11 summarises the main criteria that help the workflow analyst in determining whether a process (or a part of it) is suitable for being supported by workflow technology.

Finally, the analysis can identify the relevant reusable components, on the basis of the identified candidate workflows. Criteria could be given here for this identification, but a fair knowledge of the available patterns in the given application domain should be sufficient to list the reusable components. For example, small, specific components model exception and error handler patterns, while large application frameworks can be reused to model a whole task. In the DVD Smart Company, for example, the database errors can be treated as reusable components at the level of ‘error handlers’ or exceptions, using a library of exceptions. Instead, the Get Customer Data sub-process can be identified as a reusable application framework (see Fig. 6) because the data acquisition phase is common to various applications and constitutes a well-structured, large portion of the applications. This matter will be further clarified when illustrating the WIRES design phase, where the repository of reusable patterns is presented.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeatability</td>
<td>The process corresponds to a predictable and repeated situation, both from a scheduling and from a task assignment perspective.</td>
</tr>
<tr>
<td>Errors</td>
<td>The manual execution of the business process is error-prone, due, for example to omitted, forgotten or delayed activities or data.</td>
</tr>
<tr>
<td>Complexity</td>
<td>Scheduling and assignment criteria are complex and difficult to be executed manually.</td>
</tr>
<tr>
<td>Constraints</td>
<td>Data or authorisation constraints need to be enforced (e.g., the same actor cannot check both customer data and manipulate the credit card database, for security reasons).</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Process executions need to be monitored for the occurrence of specific events or situations that require special handling.</td>
</tr>
<tr>
<td>Tracking</td>
<td>Process tracking is a useful (or required) functionality.</td>
</tr>
<tr>
<td>Analysis and simulation</td>
<td>Process analysis and simulation capabilities offered by many WfMSs can be exploited to improve the process execution quality and optimise task assignment.</td>
</tr>
<tr>
<td>Evolution</td>
<td>When processes change frequently, the versioning and change management features of workflow tools provide better support than hard-coded implementations of the business logic.</td>
</tr>
<tr>
<td>Electronic documents management</td>
<td>Need of electronic support in document management and exchange.</td>
</tr>
<tr>
<td>Automation and Business opportunity</td>
<td>Eventually, the decision depends on whether the business value (increased revenue and reduced costs) resulting from the introduction of workflow technology compensates the costs required to develop and use the workflow application and to allow system access to the involved actors.</td>
</tr>
</tbody>
</table>

Fig. 11. WIRES ‘workflowability’ criteria.

3.4. Interaction with External Information Systems, Distribution, and Interoperability

During the analysis, we have seen that first the external systems interacting with the workflow are identified and listed as actors. Then, the interactions between the business process (or its sub-processes) are shown in the use case of zero-level and possibly detailed in the sequence diagrams. More information has to be given about the interface necessary for the interaction. Such interaction is specified as shown in Fig. 12. The External Information System (EIS) interface is specified through the EIS Name, a responsibility, composed of the main tasks of the EIS, and a collaboration denoting how the EIS is used as an UML class by the workflow. Moreover, actors that have been specified in the workflow use case are analysed to check whether they are also actors of the EIS.

In our example, the external EISs identified in the analysis are three legacy systems: Order Information System, Customer&Payment Information System, and Stock Management Information System. Moreover, we have a Shipping EIS, an Email system, and a Web system.

Some of the interactions between these systems and the Get Customer Data sub-process have been shown in the UML card of Fig. 3 and of Fig. 7. Now, we show the EIS interface for the Customer&Payment IS.
in Fig. 12. The workflow objects listed on the right-hand part of the card are basic information and data handled by the business process that have to be identified and represented by means of usual UML classes and class diagrams [26]. Only the class names and sometimes the attributes of the classes are shown, while the interaction can obviously regard other attributes to be detailed at a further refinement step.

Coming to interoperability issues, we remark that, usually, interoperability aspects are known, at least in their basic lines, from the requirements, as illustrated for the Delivery-Ship sub-process that is executed in outsourcing. In the analysis, UML sequence diagrams are used, enriched with a stereotype to represent the wait cycle of messages executed at any of the cooperating sites as a suspension of collaboration, waiting for a response that starts the cooperation again. Then, a collaboration diagram can be given to describe in detail distributed operations.

Let us give an example regarding the DVD Smart Company. Figure 13 shows the sequence diagram for the Delivery-Ship Order sub-process. The shipping service occurs via an external Transportation Company; the sequence shows the message exchange that models the cooperation. A shadowed box is used to denote the cooperating processes. A ‘start cooperation’ message and an ‘end cooperation’ message must always initiate and terminate the sequence diagram. The initiator of the cooperation must also be the terminator. The sequence of messages modelling the cooperation is depicted using boldface arrows and labels.

3.5. Summary

The main concepts and results of the WIRES analysis phase are the following:

- an overview of the business process – modelled as use cases at one or more levels, and scenarios (through sequence diagrams) for the main activities (both normal and exceptional);
- an organisational view of the process – modelled as actors and organisational units, both internal and external, through class diagrams and packages;
- a business perspective of the process - modelled as a goal hierarchy and a use case/goal matrix showing how the activities can match the business goals and opportunities fixed by the organisation;
- an opportunity evaluation of workflow technology – modelled using workflowability criteria;
- a hint regarding the possibility to reuse (portions of) the process – given by identifying a basic set of reusable elements on the basis of the business process logic represented in the use cases;
- a view of distribution aspects – modelled through interaction diagrams showing the synchronisation among distributed (networked) portions of the process
- a view of interoperability – regarding both interaction with existing legacy information systems and databases and interaction with outsourced services, through the description of the EIS interfaces.

4. Workflow Design

In workflow design, the results of the analysis phase have to be transformed into detailed workflow representations, considering the following aspects:
detailed descriptions of the tasks in the workflow based on reuse of design patterns: a model for workflow design is introduced in the following, based on an extension of UML activity diagrams, as well as a classification of workflow design patterns and mechanisms for pattern instantiation;

- specification of anticipated exceptions to the normal work: the workflow design model provides constructs to represent exceptions to the normal execution flow;

- specification of the interface with existing information systems in the organisation: interaction diagrams are used to represent the interaction with EISs and databases;

- specification of the distribution requirements: allocation of tasks and data in a distributed environment is discussed;

- specification of transactional properties in the workflow: a transaction model to associate transactional properties to tasks and groups of tasks is applied to the proposed model.

4.1. Workflow modelling

Workflow modelling at the design level is based on an extension of UML Activity Diagrams for the specification of business processes. This extension includes modelling concepts developed in the WIDE project [25] for the WIDE workflow design model. In WIRES, we prefer to adopt UML as a starting point, and to extend it with needed modelling constructs rather than proposing an alternative model. The advantage of this choice is a wider applicability of the approach and wider availability of modelling tools. However, while UML activity diagrams are a powerful tool for modelling the flow of work between activities, they do not provide constructs for modelling some relevant aspects of workflow design, such as data or temporal constraints, exceptions, and transactions.

Other authors have also proposed extensions to UML activity diagrams, but they either provide only a subset of the features actually needed for process modelling [4], or they require the use of an entirely different formalism, such as Petri Nets, to complement the UML representations [26].

In the following section we briefly describe UML 1.3 activity diagrams in order to make this paper self-contained, and then present extensions that allow the definition of some important aspects of a business process that cannot be represented with activity diagrams in their current form.

4.1.1. UML Activity Diagrams

Actions and Activities

The basic element of an activity diagram is the action node. An action represents an elementary piece of work, typically performed by a single agent. Actions are graphically depicted as a rounded box and are associated with a label that defines the name of the action (see Fig. 14(a).

Activities represent complex tasks, formed by several actions or even by other (sub-)activities. Their structure is specified separately, in a different activity diagram. Activities are useful when a process is complex and it is neither possible nor convenient to describe it in detail in a single diagram. The graphical symbol for activities is a rounded box with a small graph depicted in the bottom-right corner that denotes that the symbol is a placeholder for a complex graph.

An arrow represents the control flow between actions and activities. UML activity diagrams provide constructs that allow the specification of actions (activities) to be executed in parallel or when a given condition is verified. These constructs allow the specification of AND-split, AND-join, OR-split and OR-join in the WfMC terminology [27]. Parallel activations (AND-...
splits) are specified by means of a *synchronisation bar*. The synchronisation bar is a particular kind of node that has one input arc and several output arcs. Its semantics is that as the arc input to the bar is fired, all output arcs are fired. The synchronisation bar is also used to AND-join parallel execution flows. In this case, the bar has several input arcs and one output arc: the output arc is activated only when all input arcs have been activated.

Activity diagrams allow the specification of conditional execution of flows. Conditional flows are specified by means of a diamond with one input arc and several output arcs. An activation condition is placed on each of the output arcs. The semantics is that as the input arc is activated, the conditions associated to the output arcs are checked. Arcs whose associated condition evaluates to true are activated. This behaviour is analogous to the one defined as OR-split by the WfMC [27]. The same diamond symbol can be used to define OR-joins as well. In this case, the diamond has several input arcs and one output arc: each activation of any input arc causes the activation of the output arc.

### 4.1.2. Extensions to UML Activity Diagrams

Extensions to UML activity diagrams needed to model business processes in detail include the possibility of representing asynchronous behaviours, transactions, and multiple parallel invocations of the same action. In addition, we define the semantics of some aspects of the activity diagrams, which are left unspecified in UML 1.3.

#### Multiactions

A multiaction represents an action that can have multiple instantiations during the execution of a workflow, according to a predefined invocation precondition. The termination of the multiaction is associated to a multiaction postcondition, which specifies when the multiaction can be considered terminated. The precondition of the action specifies the number of parallel activations of instances of the same action, and the execution environment for each instance. For instance, a multiaction can allow the specification that a variable number of reviewers can be associated to an administrative action, depending on the complexity of the task and its importance, which are dynamically determined. A representation of parallel actions of this type is possible with basic UML action diagrams only if the number of parallel activations of actions of the same type is predefined. The representation of the semantics of a multiaction is given in Fig. 15, where the precondition and postcondition associated with the multiaction specify that \( i \) parallel actions have to be performed (where \( i \) is a workflow data item, whose value is determined at process execution time), and the flow can continue after terminating all the parallel actions (preconditions and postcondition can be optionally shown in the graphical representation as indicated in the figure, or omitted). Other common types of preconditions that determine the number of parallel activations include the number of available actors or the number of elements in a data structure (e.g., a vector or a list). In this case, each action is assigned to a different actor or operates on a different data item, respectively. For instance, if a workflow operates on a list of customers and we need to check the credit of all these customers, we may define a multiaction activity ‘check credit’, and specify that an action should be activated for (and operate on) each customer in the list.

Under the same semantics, we define a multi-activity as an extension of an activity.

#### Asynchronous Behaviours

UML Activity Diagrams are very powerful for modelling the normal behaviour of a process, represented by the flow of work among activities. However, process requirements often include the need for performing actions in correspondence to events that are asynchronous with respect to the normal flow of control. For instance, specific actions may need to be taken periodically or as an external event occurs. We have identified three main types of asynchronous events: data events, temporal events and external events.

- **Data events**, raised by modifications to workflow relevant data. Data manipulation operations include creation and deletion of objects or modification of attribute values.
- **Temporal events**, raised at a specified date and time, after a specified interval since a given reference event, or periodically. Temporal instants are indicated by the symbol followed by the date and time in the format \( YYYY-MM-DD HH:MM:SS \). Temporal periods are indicated by means of a notation taken from [28].
For instance, the first day of each month is denoted by 1\textit{days during months}, while Christmas is denoted by 25\textit{days during December}.

- **External events**, corresponding to notifications from external applications. Examples of external events are the cancellation of an order by a customer, a travel reservation request in a travel reservation process, a car accident in a car rental process, or a high water level in a lake water level monitoring process.

Events are represented as triggers or temporal triggers, as indicated in Fig. 14(b). Triggers may be associated to any action or activity in a UML activity diagram to indicate that specific asynchronous activation or termination conditions exist. In addition, for events for which a complex handling is needed, an activity diagram can have one associated event-specific action (or activity in the case that the event handling is complex), not connected to the normal flow of work, which represents how to deal with the event in that workflow (we denote such event-specific elements as event nodes in the following). The event indicates the starting condition for the event specific action/activity, which is executed in parallel with the normal flow. Event nodes may also include operations that imply the termination of the normal flow, or its suspension until an explicit resume operation. The semantics of such separate parts of the graph is an AND-split at the beginning of the process, including all fragments (this is not explicitly represented in the graph for readability reasons).

Event nodes are fundamental constructs for modelling exceptional situations. In particular, one class of exceptions which is gaining attention is that of expected exceptions, i.e., of situations that are part of the semantics of the process and that therefore should be modelled in the UML process description. The basic UML activity diagram is not suited for this purpose. In fact, many exceptions (such as an order cancellation request by the customer) are asynchronous with respect to the flow structure, i.e., they do not occur in correspondence of the completion of a specific action. Even when exceptions are synchronous (such as the violation of a stock reorder level) they may be caused by several different actions, and checking the constraint after the execution of each of these actions results in flow structures that are difficult to design and to maintain.

Hence, as observed by many authors (e.g., [29,30]), it is difficult to model the handling of exceptional situations by means of activities and flows. In particular, the critical aspect in process design lies in modelling the occurrence of the exceptional situation. When event nodes are available, the occurrence of the exceptional situation is modelled by event nodes, and the exception handling part, which often implies executing an action or even a process to be performed within the context of the exceptional instance, is specified by a fragment of activity diagram activated with the event node.

Several other approaches have been proposed in the literature in order to model exceptional situations. In many cases, exceptional situations have to be included explicitly in the normal flow; in this approach, however, the representation of the process may become unnecessarily cluttered with all possible exceptional situations. In addition, it is difficult to model consequences of events that occur asynchronously with respect to the process flow. Other approaches allow designers to model specific exceptional situations, such as a missed deadline for an action, or an exception handler for a generic exceptional operation, such as going back to a previously executed action or cancelling a workflow instance. Other authors [25] have proposed a rule-based mechanism for exception specification, but also this approach, although similar to the one proposed in the present paper, has the disadvantage of a representation of exceptions which is not clearly visible in the workflow specification.

**Attributes of Actions and Activities**

The graphical description of the workflow process, specified by means of extended UML activity diagrams, may be coupled with a textual description of the main characteristics of actions and activities. In Figs 16 and 17, we present the information associated to actions and activities.

The following constraints hold between the attributes of actions included in activities:

- **Input data** in an activity correspond to the **input data** of the first action(s) started in the activity.
- **Output data** in an activity correspond to the **output data** of the action(s) executed in the activity.

**Deadlines**

A frequent requirement in process modelling concerns the specification of deadlines for actions and activities, along with the specification of the actions to be performed upon deadline expiration. In order to enable the definition of these requirements, we extend UML activity diagrams with the capability of defining deadlines associated to actions or activities. A deadline may be specified as an absolute date and time, or it can be defined as the maximum elapsed time since the action started.
Transactions

In the definition of a business process, it is useful to have the description of the activities/actions that should be executed atomically or in isolation with respect to other activities/actions of the same or of different processes.

In order to specify parts of the process that need to be isolated from other actions/activities of the same or of different processes, activity diagrams are extended with the notion of business transaction (BT). When an action or activity is a BT, its intermediate results cannot be accessed by actions executing outside the BT. If another action is ready for execution but needs to access the same data, it is suspended until the BT is completed. The locking model is analogous to that of traditional database management systems: a BT acquires shared locks for all the items read by at least one action in the BT, and exclusive locks for all the data items written by at least one action in the BT. All locks are acquired at the beginning of the BT. Graphically, actions and activities with BT semantics are depicted with a thicker border, as shown in Fig. 14(b). Business transactions can be nested, in order to specify isolation requirements within a given BT.

Workflow processes may also have atomicity requirements: the designer may need to specify that an activity or an entire process should be executed in an atomic fashion. Atomicity is needed in a variety of situations: for instance, in an order process, the customer may cancel the order or, in a travel reservation process, a flight to the requested destination may not be available. In both cases, the required behaviour could be that of rolling back the order or the travel reservation process, to cancel the effects of their partial executions.
Achieving atomicity in workflow processes is more difficult than in database transactions, due to the difficulty of rolling back actions or activities. In fact, DBMSs offer mechanisms that allow to completely undo the effect of a transaction on the database. Business processes include instead actions that are performed on the real world, such as sending an email, paying a bill, or more generally accessing a legacy system or invoking an e-service. These actions cannot simply be ‘undone’. Instead, their effects have to be compensated by executing appropriate compensating actions or activities. The appropriate compensating action depends on the semantics of the process, and therefore has to be defined by the workflow designer.

When a failure occurs, rollback of a BT is performed by aborting running actions and by compensating completed actions. The BT then enters the ‘failed’ state, and the flow proceeds with the action that follows the BT in the flow. This schema is consistent with the many proposals for adding transactional properties to workflow models (see, for instance, [31–34]), and follows our guidelines of extending the semantics of UML activity diagrams with a limited and simple (but powerful) set of constructs, needed in most practical workflow projects.

**Termination**

Concerning the semantics of termination, a process termination node may be reached when some actions or activities are still in execution. This situation occurs when the process model includes non-exclusive forks that split the execution flow along several parallel threads, and the end symbol is reached in one of these threads, while the others are still in execution.

We assume that the semantics of the extended UML activity diagrams is that after the process terminates, no more actions or activities are started within the process. If a different behaviour needs to be specified during process design, such as immediately terminating running activities, or sending messages to agents that were executing such activities, this has to be explicitly specified, for instance by inserting an activity just before the end symbol that specifies the desired behaviour.

**4.2. Example of Workflow Design**

For the DVD Smart Company business process, the WF schema resulting from the design is shown in Figs 18 (top-level view) and 19 (where the activities are expanded).

The general structure of the workflow is derived from the use case diagram of Fig. 3. The ‘Get customer’s order’ sub-process represents the interaction with the customer via the email and Web services. This is decomposed in the action ‘Get Order Information’ followed by ‘Check customer already registered’. If the customer is not registered, the action ‘Get customer’s information’ is processed and the customer’s profile is registered (‘Register customer’s profile’). If the customer is registered, the action ‘Get customer’s login’ and ‘Read customer’s information and preferences’ are processed. ‘Check order’ is represented as an activity since it can be decomposed into ‘Validate order’ and ‘Check credit card valid number and expiration date’, as shown in Fig. 20(a). Figure 18 indicates two possible decisions after the order is checked: terminating the WF if the order is not correct, continuing the WF otherwise. Both scenarios can be represented as part of the normal flow, since they are part of a normal possible evolution of the flow.

Then, the Process Order use case is decomposed into Satisfy Order Order Inquiry, and Wait for Delivery (this last sub-process is executed only if the order is not complete). Satisfy Order appears as an activity since it can be further decomposed, as shown in Fig. 20(b). First, Check item stock level is specified, then Broadcast distributed order follows if the stock level is not correct, Allocate from stock otherwise. In Fig. 19, Satisfy Order also appears as a multi-activity since several activities must be done in parallel to process all the rows constituting the order ($i = \text{number of rows}$); the activity
terminates when all the rows have been processed ($j=\text{number of rows}$). Finally the last sub-process taken from the use case diagram, Delivery-Ship Order, is decomposed into two parallel sets of actions aiming at sending a Request for shipping on one hand and Send email confirmation and Send payment receipt on the other hand.

Concerning exceptions, different exceptions are associated to the WF schema to handle situations where there is a need to consider the occurrence of specific events. To handle authentication errors and database errors, the AuthenticationFailure and DatabaseFailure exceptions are used to notify the responsible (See Fig. 4). Time-related exceptions,
PwdTimeout and CertificateTimeout, are also used to model the constraints given in Figs 6 and 7 with respect to the delay allowed for a password to be modified and for a security certificate to be sent. Finally an exception, Re-Order, is also used in the Process Order activity to re-order an item if the Allocate-from Stock task leads to a low stock level.

4.3. Patterns for Workflow Design

In WIRES, we assume that UML diagrams are not designed from scratch, but rather by assembling design components.

A pattern catalogue must be available, to provide the building blocks for designing workflows. We assume that many processes can reuse design artefacts from previous designs, as proposed in the software design literature [35]. In the process design area, design by components is starting to emerge in recent proposals [19, 26, 36]. In this paper, we assume that components are organised in a pattern catalogue [19].

In the catalogue, patterns are artefacts that are generic and may be reused with an instantiation operation. Each pattern is generic, since some of its parts are parametric. With instantiation, a workflow design component is generated from the pattern, which can be included in a specific workflow design. Pattern instantiation is performed based on rewriting rules and selections between alternatives. Rewriting rules allow the transformation of patterns containing generic objects. Each generic object contained in the pattern must be transformed into a specific object. For instance, in the pattern ‘send a <reminder> to <somebody>’, the generic objects <reminder> and <somebody> must be transformed in a design component where the typology of the reminder and of the destination are specified. Alternatives may be defined in a pattern to indicate design alternatives to satisfy a given requirement. For instance, an official notification can be performed in person by public officers or via secure email. In a given process, one of the alternatives may be chosen, or both, according to the non-functional requirements associated with the workflow.

Patterns are usually instantiated at design time: the designer selects the appropriate patterns, guided by suggestions about reuse gathered in the analysis phase and by searching the pattern repository during the design phase. Some of the patterns, however, are useful for defining services which may be invoked dynamically; in this case, the pattern is not instantiated at design time, but rather provides a framework for dynamic activity invocation. Details about realization of workflows with dynamically invoked services are discussed in the following section on the implementation phase and in the concluding remarks.

Patterns are classified into three categories:

- built-in patterns: represent basic situations independent of the semantics of possible applications. This group includes patterns for basic exceptions, such as a temporal delay handler, for authorisation of agents or operations, for workflow starting and termination situations. In the DVD Smart Company example, an Authentication Failure pattern is used in the Get Customer’s login activity.
- generic tasks: patterns in this group allow the definition of tasks with specific characteristics, including possible exceptions associated with these tasks. Exceptions can also imply the invocation of an
isolated task to perform part of the exception handling associated with the task. For instance, the Get customer’s information activity is an example of use of a design pattern in the example presented in Fig. 19; timeouts may be associated to this task as an exception (instantiated in this case in PwdTimeOut and CertificateTimeOut). An example of pattern and its instantiation is shown in Fig. 21.

- generic application oriented structures: this group contains fragments of workflows. A classification of several types of fragments is given in Casati et al. [37], e.g., as design patterns in Fowler [38], and control patterns in van der Aalst [39]. For instance, in the DVD Smart Company order, the initial phase of getting customer order information follows a generic pattern, which is personalised in the application being designed by checking the customer’s login and preferences if the customer is already registered.

4.4. Designing Interactions with External Information Systems and Applications

The issue of interfacing the EIS and the workflow is dealt with during the design. The UML EIS Interface is incrementally specified to detail the interaction between the EIS and the workflow. Such interaction is specified by giving four interaction modes (called interaction points) that allow the EIS to collaborate with the workflow:

- **Reaction points**: these define the set of workflow events that have to be sent to the EIS to collaborate with an EIS procedure or with an external application.
- **Data insert/cancel/convert points**: these are methods that allow the workflow to send/receive data to/from the EIS and to execute the conversion between EIS data and workflow data. Such conversion is done by an EIS Interface class in order to achieve compatibility between the EIS data.
- **Call points**: these are entry points to the EIS procedures or external applications that can be invoked from the workflow whenever needed, for example to acquire data from an EIS database.
- **Exception points**: these are exceptions rising in a workflow task that require the invocation of an EIS procedure. For example, in the exceptional scenario ‘Cost not approved’, a procedure of an EIS can be notified that a cost has not been approved and therefore the Cost database of the EIS has to be updated.

An interaction diagram is given describing how groups of EIS objects collaborate with the workflow, in terms of sequence diagrams to represent the EIS Interface during the interaction with the workflow (See Figs 6 and 7). The diagram helps the workflow designers to identify the interaction points between the workflow evolution and the EIS evolution and the points of data/control exchange.

The output of the design of EIS is the design specification of a wrapper module incorporating the EIS and the external applications, giving an EIS Interface class designed for collaboration between EIS applications and the workflow.

In the design phase, data conversion should be specified regarding the main data exchanged between the EIS and the workflow. In particular, this can be done by analysing the sample scenarios provided for the relevant workflow sub-processes. For example, consider the scenario of the Get customer data sub-process (see Fig. 7), interacting with two EISs: Customer & Payment and Web & E-mail. Upon an exceptional situation, e.g., ‘Customer does not exist’, the workflow variable concerning the customer (identifier, name, address, …) must be translated into an appropriate set of attributes according to the format of the Customer & Payment and Web & E-mail systems.

4.5. Designing Workflow Distribution and Interoperability

Workflow design has to define as far as possible the issues related to process distribution over several WfMSs. In fact, it is often not possible or not convenient
to support the execution of an entire process with a single workflow engine. For instance, consider the process of hiring a new employee in a research lab. This process includes a part related to the preparation of the office, office supplies, and suitable computer and software for the new researcher, and a part concerned with financial and administrative issues. The first part involves decisions and actions to be taken at the lab site, while agents of the financial department execute the second. While it is feasible to have a single WfMS located in one department executing the process and dispatching work activities to the remote department when needed, this solution causes a considerable communication overhead. A more efficient approach might consist in exploiting two WfMSs to execute the process: one located at the lab site, and the other located at the financial department. In other cases, the interaction of distinct processes of different organisations could be needed to provide an external service. Examples of this need is providing one-stop shops in public administrations, where several organisations cooperate to provide a given service, which is perceived as a whole by the citizen requesting it, such as the registration of a new company, but implies complex procedures internally involving different administrations. In this case, each administration may have its own WfMS, and different processes have to be integrated to provide the service.

At the design level, distribution design has to:

- identify the process fragments which present homogeneous characteristics in terms of access in the organisation to activities belonging to the fragment
- evaluate an ‘optimal’ allocation of the identified fragments of the distributed process, with respect to given evaluation parameters.

In the literature, while some attention has been given to evaluating organisational costs of distributed architectures [40], little attention has been given to the design of distributed processes. A systematic approach to partitioning processes has been proposed within the MENTOR project [41], which defined an algorithm for partitioning a centralised state and activity chart into a provably equivalent one; the evaluation of the result is performed in terms of number and size of the synchronisation messages needed between the partitions. Other research work (e.g., within the CrossFlow ESPRIT project [42]), tackles the issue of controlling processes across different WfMS in terms of synchronisation messages and definition of transactional properties.

The problem of designing a distributed process architecture is still a research issue.

The design of the process distribution can be performed in the following steps:

### Step 1: Identification of Task Execution Location

The first step consists in identifying the physical location where each task is executed. This information can be typically obtained by determining the roles to which the activity can be assigned and the groups or functions to which these roles belong. Groups or functions are often related to the physical location. If a task can be executed in different locations, by persons in the same role at different locations, the task is split in a number of equivalent tasks, differentiated only for the execution location. For instance, in our case study, the activity ‘broadcast distributed order’ can be executed in different locations and is therefore split in different equivalent tasks (for each retail site supporting the distributed service).

### Step 2: Identification of Process Blocks

In this step the designer identifies groups of tasks executed at the same location. According to the methodology developed by Muth et al. [41], the process has to be partitioned into a set of component workflows (blocks) and coordination messages. The problem to be solved in this phase is the definition of coordination messages which are sufficient to execute the distributed process equivalently to the centralised one, but minimal in the sense that only information useful for task execution is transmitted and synchronisation messages are sent only to the blocks for which they are relevant. The minimal information and coordination overload has to be identified.

### Step 3: Block Allocation

Block allocation involves weighting the advantages and disadvantages of a distributed process execution with respect to a centralised one. Following the allocation methodology proposed for distributed database design in Ceri et al. [43], several block allocations can be evaluated in terms of cost of the allocation (benefits are evaluated in terms of reduced costs). Different alternatives for block allocation are evaluated, and their cost compared. Allocation can be non-redundant or redundant. Non-redundant allocation is considered first, allocating each fragment to the location that has minimal costs with respect to the chosen parameters. Redundant allocation allows the execution of the same task in different WfMS. The advantage of allocating the same task to several WfMS is an increased locality of activity execution, in particular when external ISs are invoked, and increased performance due to a better resource exploitation. In terms of costs, an evaluation of organisational costs and technological costs has to be
performed. The organisational costs can be evaluated following the indications of Francalanci and Piuri [40], while the technological costs have to take into account costs for local execution and transmission costs for the information and synchronisation messages to be sent between partitions. Technological costs include an evaluation of the execution time.

**Step 4: Workflow reconstruction**

The final step consists in assembling the fragments allocated to a WfMS, in order to reconstruct the processes performed locally, and defining the interfaces with processes executed by external WfMSs.

**5. Mapping to Implementation**

This section discusses how to implement the workflows specified in the design phase on top of a WfMS. Workflow implementation is centred around two main issues: (1) selection of the WfMS and application integration solution; and (2) definition of the designed workflows in the language supported by the WfMS. We next provide guidelines to address these issues.

**5.1. Selection of the Target Workflow Management System**

Once the workflow design phase has been completed and the characteristics of the process have been identified and specified, a suitable WfMS must be selected for the implementation.

While most WfMSs share the same basic concepts and provide similar workflow models, they do have significant differences that must be considered in the choice of the appropriate system. Indeed, it has been shown that an early and inappropriate WfMS selection may lead to considerable difficulties in the implementation and enactment of the workflow processes [20].

Several criteria have to be considered in the selection of the WfMS. Some of them, such as cost, robustness, performance, and existing business relationships with the vendor, are common to any application, and therefore are not discussed here. Note that our goal in WIRES is not to survey commercial WfMSs and compare their features. Instead, we are concerned with selection criteria that are peculiar to workflow management. In particular, our aim in this section is to help the project team in identifying the crucial characteristics of a workflow application with respect to system selection, in order to then be able to look for a WfMS that can provide these features.

In the following we list the characteristics and possible requirements of the workflow application that should be considered to address WfMS selection. For each of them we also discuss the drawbacks resulting from the selection of a WfMS that does not provide that feature, to help the development team in weighting the trade-offs that are part of every selection.

**5.1.1. Degree of Automation**

Workflow applications differ in the degree of automation. At one extreme, a workflow may only involve tasks executed by human actors, typically by filling out forms or by writing documents. These are often referred to as *human-oriented* workflows [12]. WfMSs that support this kind of process offer simple and easy-to-use process definition languages, toolkits for designing Web interfaces, and integration with email systems and document-processing applications. At the other extreme, a workflow may be composed of tasks that perform transactions over heterogeneous and autonomous systems, possibly also invoking services performed by different enterprises. These are typically referred to as *system-oriented* workflows. WfMSs supporting system-oriented workflows enable and simplify the interaction with workflow-invoked applications. They are typically integrated with Enterprise Application Integration (EAI) tools, to access back-end applications such as databases and ERP systems and to perform e-business transactions across enterprises according to some B2B standard. In general, a given application will sit in between the two extremes, involving both manual and automated activities. At this stage, it is important to characterise the number and type of external applications involved in workflow execution, in order to evaluate the effort in developing adapters to interface with them and to investigate the availability of existing EAI or workflow products that already include such adapters. WfMSs designed for supporting human-oriented processes are suitable if the number of manual activities is much larger than the number of automated ones, and if the interaction with external applications is limited to a few operations which do not change frequently over time. Since this is often the most important aspect in WfMS selection for system-oriented applications, at the end of this subsection we include a discussion about implementation of EAI solutions, in order to guide the reader in understanding whether there is the need for a WfMS that includes an EAI platform. In our reference example, the workflow needs to access several enterprise applications as well as e-services. Integration with EAI tools will therefore simplify the development and maintenance of the workflow application.
Drawbacks of inappropriate selection: all WfMSs provide some support for human-oriented applications. In general, selecting a system-oriented WfMS for human-oriented applications does not provide major drawbacks, although it can result in slightly longer development time or less attractive and customisable Web interfaces. On the other hand, selecting a human-oriented workflow for a system-oriented application would most likely result in very long development time and large effort for maintaining the workflow application. In fact, human-oriented workflows do not typically include the appropriate infrastructure to communicate with back-end applications. The development of such infrastructure is a tedious and very labour-intensive task. In addition, WfMSs supporting system-oriented processes typically have the robustness, availability and performance requirements that meet the needs of high-volume, mission-critical applications.

5.1.2. Need for Adaptive and Dynamic Behaviours

Depending on the business and technical environment to which they belong, workflows may have different adaptive and dynamic requirements (see [44] for details on adaptive and dynamic flows). A workflow is adaptive if it can adapt to changes in the environment and to the characteristics and needs of each individual instance with minimal or no user intervention. For example, two adaptive features that are often needed in workflow applications are the multiaction and the late binding of nodes to actions and actors. Multiactions have already been introduced in previous sections. Late binding refers to the ability of determining at run-time the actor and/or the exact task to be performed, according to actions defined in the UML diagrams. For instance, in the design of the workflow for our reference example, we may specify that a transportation service is needed at some given point in the process. However, information about the exact type of transportation needed (e.g., by train or by truck) and the suitable service provider may vary on an instance-by-instance basis, and therefore may not be available at process definition time. A mechanism of late binding is suitable to implement dynamic service identification. In fact, a set of specifications (e.g., expressed in XML) can be processed by the WfMS to identify the service that better fits the needs of the process at any given point. The specifications must be enriched with elements that allow the WfMS to choose the transportation means or the retail site: such elements are attributes of services in our approach (see [8], which specifies which service is available at which site and under which cost and performance parameters). Moreover, these attributes of service descriptors specify what alternative service is available when a service provider (e.g., a sell point) is unreachable. This alternative is given by means of fuzzy elements that describe ‘how well’ a service fulfills the required functionality (e.g., the delivery of goods) because it is ‘similar’ to the needed service. The similarity can be computed using affinity and vicinity criteria of syntactic and semantic type. We refer the interest reader to Castano et al. [45] for details about similarity measures. Late binding of nodes enables processes to adapt to the needs of different customers and execution needs and constraints, by determining the actor and action to be performed in a given node only at execution time.

Workflow applications also differ in the need for dynamic changes, i.e., for modification to the flow while they are in execution. There exist two types of dynamic flow modifications that can be required, ad hoc and bulk:

- **Ad-hoc changes** are modifications applied to a single running workflow instance, and are typically needed to manage ad hoc requests from customers or to cope with unexpected situations that prevent the progress of the normal execution.
- **Bulk changes** refer to modifications collectively applied to a subset of (or to all) the running instances of a workflow. They are needed, for instance, when a new law requires different procedures, and every newly started instance as well as running ones should be modified to follow the new directives.

Administrative workflows executed in government offices are most likely to be rather static, since the flow of execution is derived by the law. Instead, workflows that support e-business applications, such as our DVD example, are executed in a more dynamic environment and must adapt to the needs of each individual customer or to service unavailability, thereby requiring adaptive features and support for change. Still, a description of provided services, enriched with attributes describing the relevant features of a service, prove useful to dynamically adapt the workflow execution to run-time requirements and constraints. It is the responsibility of the process analyst to understand the characteristics of the business and IT environment in which the workflows will be executed and identify the need for dynamic changes, along with their frequency and extent.

*Drawbacks of inappropriate selection:* in general it is very difficult to support workflows with adaptive and dynamic requirements if the WfMS does not provide help. While some process-mapping techniques can be applied to simulate multiaction nodes and other adaptive features (as discussed in [39]), simulating late-binding nodes and dynamic changes features would result in huge implementation and process maintenance efforts.
Hence, the project team should definitely search for a WfMS that supports adaptive and/or dynamic features if such are the needs of the application.

Unfortunately, the large majority of WfMSs can only manage static and non-adaptive flows. This is due to the fact that only recently WfMS users are requiring dynamic and adaptive behaviours to meet the needs of Web-based applications. The only adaptive feature supported by most systems lies in the dynamic selection of the actors to which tasks should be assigned for execution. Dynamic workflow evolution features are only provided by a handful of systems, and are limited to allowing the re-execution of parts of the flow for a specific instance or the manual re-assignments of tasks to actors.

### 5.1.3. Need for Exception Support

Different workflows may have different requirements in terms of exception support. These range from no support needed, to deadline management, support for synchronous exception (i.e., errors in action execution), and support for asynchronous exceptions (i.e., ability to detect and react to events that occur asynchronously with respect to the execution flow). The need for exception may be determined by looking at the number and type of triggers defined in the designed workflows. In particular, the need for asynchronous exceptions may be determined by considering the number of actions that do not correspond to work to be done, but only wait for an event to terminate and activate the successor task. For example, the DVD example includes several synchronous and temporal triggers. Hence, a WfMS supporting triggers is desirable.

**Drawbacks of inappropriate selection:** while support for exception is important, there are several ways in which exceptional behaviours can be modelled on top of systems that do not provide explicit support. Although this would result in a slightly increased modelling effort, it can be supported by patterns and guidelines for workflow design, coupled with the development of simple, event-polling external applications, as discussed elsewhere [19,46].

### 5.1.4. Need for Time-Controlled Activation

Some workflow applications may require processes to be instantiated at a specified time instant, or periodically. For instance, month-end reconciliation activities are typically started at the beginning of each month. Time-controlled activation may be also needed within a process instance execution, when a task should only be activated at or after a specified timestamp, periodically, or after a given interval has elapsed since the completion of a previous task. The need for time-controlled activation of processes can be determined by looking at the process attributes. Instead, the need for time-controlled activation of tasks can be determined by considering the number of actions that do not correspond to work to be done, but only wait for a time event to occur, so to terminate and activate the successor task. The DVD case study does not have this kind of requirement.

**Drawbacks of inappropriate selection:** analogous considerations to those presented for exception support apply here. In fact, timer nodes can be implemented with limited effort, as discussed in Casati and Pozzi [46].

### 5.1.5. Need for Interoperability and Support for Standards

Workflows differ in the need for interoperability with other workflows and for B2B interactions. Traditionally, workflows have been designed to be executed within an enterprise. Recently, however, workflow applications are increasingly used to support e-commerce transactions and conversations among businesses. B2B interactions typically occur in the form of XML message exchanges, and often according to standards such as CBL, BizTalk or RosettaNet. Our example, although simplified, does include interactions with external e-services, and therefore calls for B2B integration support, especially if no EAI tool with B2B capabilities is used.

**Drawbacks of inappropriate selection:** the development of this kind of workflow application is considerably simplified if the selected WfMS provide support for established B2B standards and if the WfMS vendor is committed to keep supporting emerging standards that will most likely cover many areas of the B2B space. There are currently several vendors on the market that support the most successful B2B standards. In particular, almost all providers of system-oriented WfMSs now support RosettaNet-based conversations. However, if no support is provided by the system, this can be added by implementing a B2B adapter or by integrating an existing XML adapter. The implementation effort is limited, and is therefore typically tolerable.

### 5.1.6. Task Assignment Requirements

With respect to task assignment modes and organisation models, workflow applications can be divided into two categories, according to whether they need to assign tasks by relying on an external application or not. In the first case, the workflow application contacts an external resource manager each time a task is scheduled for

---

2A classification of the different type of exceptions is provided in Casati and Pozzi [46].
execution. The resource manager will return a reference to the appropriate actor, to which the WfMS will then deliver the task. This approach is useful in situations where the structure of an organisation or of the available applications is maintained outside the WfMS. For instance, a workflow may need to assign tasks depending on the employee database, or may invoke e-services based on the content of directories, Web-based advertisement services or matchmaking engines. At the opposite side of the spectrum are applications that do not need to rely on external applications for task assignments (typically because the actor base is rather static and the workflow assignment requirements are orthogonal with respect to the information stored in enterprise employee databases or other directories), as is the case of our DVD example. In these cases, workflow development would be simplified if the WfMS could offer a built-in facility for actor management and task assignments, based on the characteristics of the different actors and their role or level in the organisation.

**Drawbacks of inappropriate selection:** although a few WfMSs do not provide a built-in organisation meta-model and an actor repository, its implementation is relatively simple if the situation so requires. Indeed, such WfMSs typically provide adapters to access databases and repositories on top of which an organisation meta-model can be defined, instantiated and queried. The opposite situation, i.e., the case in which the workflow application requires an external resource manager but the WfMS assigns through the internal actor repository, presents more challenges. In fact, in this case the implementers are faced with several alternative options (see Fig. 22):

1. The WfMS administrator keeps the actor base consistent with the external repositories of interest. This alternative is viable only if the number and type of repositories of interest are small and constant over time, and the size of these repositories is limited. Otherwise, the need for implementing and maintaining multiple interfaces and for loading large amounts of data would impose excessive burdens in terms of development and maintenance effort. In addition, this option is viable only if the actor selection language offered by the WfMS is powerful enough to express the desired actor selection semantics.

2. To precede every task that needs to be externally assigned with a task whose purpose is the actor selection. In addition, this approach requires the implementation of interfaces with the actor repositories.

3. The tasks are always assigned to an external application that looks for the resource and then delivers the task to the actor or application. Like the above approaches, this one also requires the implementation of interfaces with the actor repositories. In addition, it has the drawback of bypassing the worklist server (thereby inhibiting monitoring, tracking and analysis functionalities), since tasks are assigned directly by the actor selection application. Despite its drawbacks, this alternative is the most commonly applied in practice, in particular in the service composition space, to access e-services by interfacing with e-services platforms. In addition, the implementers may be forced to follow this path in cases where the actors need the work to be pushed to them (most WfMSs work with a pull model) or when the actor base is so large and dynamic that it is impossible to maintain worklists for each user.

### 5.1.7. Need for Business Transactions

Traditional applications did not require transactional semantics, and this is why most WfMSs did not include transactional support at the workflow level (until very recently). However, the need for business transactions is growing, especially in workflow applications that support e-business transactions. The need for transactional support can be evaluated by looking at the number of business transactions defined in the flows specified during the design phase. For example, in the DVD case study the workflow activity ‘Process Order’ requires transactional semantics.

**Drawbacks of inappropriate selection:** similar considerations to those discussed for exception support are valid for business transactions as well. In fact, there are ways in which business transactions can be implemented on top of systems that do not provide this functionality, as discussed in Alonso et al. [32]. While integrated transaction support is certainly a benefit and reduces the implementation effort, this is in general less crucial with respect to issues such as degree of automation or need for adaptive behaviours.

However, the solution proposed in Alonso et al. [32] is viable only if the business transaction is relatively simple and does not include complex flow-routing behaviours and parallel execution of activities.

### 5.1.8. Understanding EAI Requirements

The implementation of the client interfaces is typically the most time-consuming part in workflow application development for workflows where interaction with EISs is needed in the workflow. Indeed, if a workflow only involves human actors, then the features offered by most WfMSs, in terms of automatic web page generation and link to email systems, can make the development of
client interfaces fast and efficient, especially if the ‘standard’ web forms automatically generated by the WfMS are acceptable. Otherwise, custom forms may have to be defined and linked to the workflow data. However, the use of custom forms is also supported by most WfMSs, so that this task is made relatively easy as well. On the other hand, the development of system-oriented workflows may involve a large design, development, and testing effort, depending on the number and type of systems to be invoked. This is a typical EAI problem, and therefore has the same complexity and design choices of such problems. The workflow implementors are faced with two main options about how to integrate applications. A first option consists in developing an ad hoc interface for every invoked system (see Fig. 23). Tasks to be executed by these systems are assigned to the corresponding adapter instead. The adapter periodically connects to the WfMS, retrieves the work item, converts data into the appropriate format and invokes the corresponding API.

Fig. 22. Different options for the assignment of workflow tasks: by accessing a local repository (a), by using an actor-selection task (b), or by relying on an external broker (c).
operation on the application. Then, the results are converted back into the format accepted by the WfMS and the work item is returned.

The alternative approach is to leverage EAI technology for the integration, as shown in Fig. 24. In this case there is no need for developing adapters, since these are provided by the EAI vendor (the WfMS typically provides the WfMS-EAI adapter). The development work in this case consists in configuring the adapters in the (often user-friendly, high level) adapter configuration language.

As usual, the choice of do-it-yourself versus using an EAI tool is determined by a Return On Investment (ROI) comparison of the two approaches. The comparison can be made by following a scheme similar to the one shown in Fig. 25, where the qualitative comments must be replaced by the estimated costs.

As the figure describes, the ad-hoc alternative has the following drawbacks:

- High development costs, for two reasons: (1) developing many adapters from scratch implies several man-months of design and development time; and (2) developers are required to be highly
skilled (and therefore highly paid), since this task is more complex with respect to configuring adapters. In particular, development costs can be very high if many different systems need to be integrated, and if their interfaces are very complex in terms of number of function calls and parameters. Older applications may not even provide an API, and require terminal emulation.

- **High opportunity costs**, i.e., delay in the availability of the solution that results in missed profit, since the old, non-WfMS-supported process remains in place for a longer time.
- **High maintenance costs**, since adapters need to be modified due to changes in the workflow requirements or in the external applications.

On the other hand, the EAI-based alternative dramatically reduces the above costs, at the expense of high software licence, hardware and training costs, due to the acquisition of the EAI platform.

<table>
<thead>
<tr>
<th>COSTS</th>
<th>Do-it-yourself</th>
<th>EAI-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software and hardware costs</td>
<td>Zero or very low costs in most cases</td>
<td>Costs due to EAI licence, hardware, software, installation, training (incremental costs compared to simply acquiring the WfMS without the EAI platform)</td>
</tr>
<tr>
<td>Development costs</td>
<td>High design, coding, and testing costs due to long development time and high costs of software developer</td>
<td>Low costs, due to short development time and low costs for developers (fewer skills are required)</td>
</tr>
<tr>
<td>(development time x monthly costs of developers)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opportunity costs</td>
<td>High opportunity costs, since development time is longer</td>
<td>Low opportunity costs, since development time is shorter</td>
</tr>
<tr>
<td>Maintenance costs</td>
<td>High maintenance costs, since adapters need to be maintained and must evolve with the characteristics of the workflow application and of the legacy system</td>
<td>Low maintenance costs, since adapters are developed by the EAI and the WfMS vendor</td>
</tr>
</tbody>
</table>

**Fig. 25.** Cost comparison for the selection of the application integration strategy.

5.2. Mapping Activity Diagrams into Workflow Specifications

Once the workflows have been designed and the WfMS has been selected, the actual implementation effort can start. This involves mapping UML activity diagrams into workflow specifications (in the language supported by the selected WfMS) and developing or configuring adapters and custom forms. While a thorough design does contribute to making this operation relatively smooth, mapping activity diagrams into workflows is not a matter of simple syntactical translation between formalisms. In fact:

- The design phase typically aims at modelling a process in detail. However, when implementing them on top of a workflow platform there are performance and usability issues that might suggest minor changes to the process, such as merging or splitting of activities. These issues sometimes depend on the characteristics of the selected platform.
- The workflow language could lack the expressive power required to model some of the (extended) activity diagram constructs. In fact, it may happen that no system satisfies all the requirements of a given workflow application. Hence, some functionality could be lacking and must be ‘simulated’, as discussed below. On the other side, sometimes the workflow language includes some advanced feature that can be leveraged in order to model a specific process semantics.

5.2.1. Implementing Activity Diagrams

When faced with the actual implementation, the designer must handle a number of detailed issues that are typically overlooked in the previous phases (also because they depend in part on the selected WfMS). For this reason, even if the workflow language supports all the features of (extended) UML activity diagrams, the mapping is not straightforward.
One of the main differences between the designed and the implemented workflow can be the granularity of actions. In the design phase, action granularity is typically driven by semantic considerations, so that logically distinct tasks are typically represented by separate actions. In a workflow implementation, the granularity is typically driven by the characteristics of the workflow actors, of the assignment criteria, and by the performance and tracking requirements. These criteria may require the merging or splitting of actions. A sequence of two actions is typically merged into one when the actor executing the two is the same. For instance, suppose that our reference workflow requires the approval of an order. If the same person performs the two tasks, then having two separate actions would require this actor to log in to the system twice in succession. This is annoying for the user and increases the load on the system (since the workflow engine must process two completion messages and must schedule twice instead of once). Another reason for merging two actions could be imposed by the characteristics of the actor: for instance, in our reference example, an automated system could offer a function capable of both checking a purchase order and of sending it to the delivery service. Hence, even if the two functions are considered as being logically separated by the designer, they have to be implemented within one node.

In other cases it might instead be necessary to split an action into parallel or sequential activations. A possible motivation could again be driven by the characteristics of the actors. In fact, an action may actually require the invocation of several actors. For instance, the action ‘Read customer’s information and preferences’ may involve access to several databases, depending on where the information is kept and on how it is structured. In addition, the invoked application may not offer a single operation that fulfils the purpose of the action node. Instead, it may require a sequence of method invocations. A further reason could be to expose more business logic at the workflow level, and to manage/track workflow execution with more detail. In fact, while decisions about business logic and monitoring needs from a business viewpoint are defined at design time, the implementation may reveal the need for tracking and monitoring more details than originally planned. For instance, the transactional and recovery features of an invoked application may require that a node is split into a sequence of two separate invocations, to allow for monitoring and logging each intermediate step and therefore having the information needed to enable a more efficient failure recovery.

Typically, activities in UML activity diagrams are mapped into sub-processes. However, there may be cases in which this is not possible or not recommended. For instance, the majority of WfMSs suffer from performance problems when activating a sub-process. In particular, the sub-process activation phase can easily last 1 minute. This is unacceptable for e-commerce transactions where the user (or even the employee) could be waiting on-line. Besides increasing latency, invocation and execution of sub-processes also reduce throughput and consumes many WfMS resources (especially in terms of main memory), thereby affecting the overall system performance. Hence, the implementers must consider the options of expanding sub-processes into the calling workflow, depending on the characteristics of the WfMS and hardware system at hand, on the forecasted system load, on the time-related requirements of the application, and on the reusability of the sub-process definition.

In contrast, it is sometimes necessary to follow the reverse process, and transforming a part of a workflow in a sub-process. This can happen in cases where the WfMS repository already includes workflows that execute the same business logic, and which are invoked by other processes as well.

Finally, another reason for modifying the UML activity diagram specifications may be due to specific error-handling needs for some invoked applications and to the characteristics of the selected WfMS. For instance, it is possible that the WfMS does not support retries in cases where the application does not respond to or acknowledge the request. In this case, it may be needed to handle ‘retries’ at the workflow level, by modelling appropriate nodes, paths and timeouts.

5.2.2. Filling the Gap Between Activity Diagrams and Workflow Languages

The second mapping problem we consider is that of mapping features of activity diagrams that are not supported by the workflow language at hand. Several researchers have already addressed various aspects of this problem (see, e.g., [32,39,46]). In particular, van der Aalst et al. [39] focus on mapping the process flow, Casati and Pozzi [46] show how to deal with exceptions and asynchronous events, while Alonso et al. [32] address the problem of mapping business transactions. In this section, in order to make this paper self-contained, we will just describe the main concepts and principles behind the mapping techniques proposed in these papers. The interested reader is referred to the mentioned papers for details.

In general, there are two options for mapping constructs that are not supported by the workflow language:
Define additional nodes and arcs in the process flow in order to implement (or at least approximate) the required semantics.

Develop an ad hoc client application that can provide the desired behaviour.

The two approaches can be combined. For example, consider the problem of implementing a multi-action node on top of a system that does not have this capability. In simple cases, where the maximum number of parallel activations is known, the multi-action can be simulated by a structure such as the one depicted in Fig. 26. The multi-action is implemented as a set of parallel action nodes, whose cardinality is equal to the maximum number of parallel activations. At run-time, each node in the set is executed or skipped depending on the number of parallel instances actually needed. In the DVD order processing example, the condition can be based on the number of rows in an order (Fig. 26b). Another solution could be to put the action node inside a cycle and set the loop condition in such a way that the appropriate number of instances is executed (Fig. 26c). As the figure shows, in this second mapping alternative the workflow model remains simple and manageable, and there is no need to know in advance the maximum possible number of parallel activations. However, these advantages are gained at the price of losing parallelism, since the invocations are done in the appropriate number, but in sequence. Therefore, the applicability of this second mapping solution depends on whether the requirements of the multi-action were dictated only by the fact that the number of instances is unknown at design time, or also by the need of parallel activations.

On the other hand, the external application solution could, for example, be adopted when there is the need of implementing action nodes that only wait for a specified time interval before completing. In this case, the workflow developer could implement a function that waits for a specified time interval, and then returns. When the waiting behaviour is required in a workflow specification, the designer can define a ‘wait’ node that calls this function, providing the interval as input parameter.

Both approaches have disadvantages: The flow modification approach may result in ‘spaghetti’ workflows that are difficult to specify and even more difficult

---

**Fig. 26.** A multi-action (a) and two possible mapping strategies. Implementation of exclusive splits on top of systems that only provide exclusive or parallel splits (b), and iterative execution of the same action node (c).
to maintain. The external application approach requires a coding effort, and may result in business logic executed outside the workflow, thereby limiting the usefulness of the WfMS monitoring and tracking tools. However, in many cases these mapping approaches can fill the semantic gap between activity diagrams and workflow languages.

6. Concluding Remarks and Future Work

This paper has described a methodological approach to the design of workflows and has provided examples of application of the methodology to the development of distributed application. The resulting WIRES methodology covers the analysis of business processes and their workflowability, the design of advanced interoperable applications, and the mapping to commercial WfMS packages under a unified modelling paradigm based on UML. Some extensions to UML are proposed to cover the organisation perspective during the analysis, and to treat transaction and exception handling during the design. Reuse of workflow components is a featuring issue of WIRES.

Under the WIRES approach, it may become easier for the developer to systematically produce a working application since interoperability, distribution, interaction with existing information systems and non-functional requirements of the workflow are taken into account since the analysis.

When designed workflows have to be mapped onto a commercial workflow product, WIRES provides criteria for selecting the WfMS that ‘best fits’ the rich modelling concepts provided by WIRES at the analysis and design stages. At implementation time, many choices are still left by WIRES to the implementers, since the task of obtaining a working distributed application which can be executed by a WfMS, starting from the identified features, depends much on other criteria such as efficiency, task assignment requirements, compliance with standards, and cost motivations.

Coming to comments on reuse of components, the WIRES analysis phase identifies workflow frames that model common situations in different domains. Examples are data management components, or usual procedures and related agents, e.g., for checking the availability of goods from stock. The design phase then selects the workflow patterns from a library of reusable components and provides guidelines to personalise these components. We outline that one difficulty emerged from our research is the possibility to early identify suitable components to be reused in the application, such as Get Customer’s Data in an order processing service.

Another key aspect regards the granularity level of the reusable component: large components can be difficult to tailor and adapt, while the drawback of small components is their scarce contribution to soften the development process. Examples of fine-grained components are database or workflow exceptions (treated as triggers in WIRES), while sample large application frames are the Check Order Availability procedure that, through proper parameter setting, can be reused to cover a large portion of the application, in the style of what software components do in current object-oriented development methods based on reuse.

To face the reuse aspect, in previous work [18] we have developed a library of reusable components at different granularity levels: from application frames, to basic triggers for managing exceptional events. In another line of research, we have studied and applied techniques for selecting reuse candidates based on criteria of ranked, best-fitting frameworks [45]. The retrieval criteria and mechanisms turn out to be most relevant for the success of the WIRES approach, since the analysis relies on tools for inspecting the available components and for identifying reusable candidates. Elsewhere [8,45] we have described the way to query a reuse repository to ‘select the component that is closer – or most similar – to the needed functional and non-functional features’ of the current application. Such a query tool really makes a difference in efficacy of the method of development under reuse.

Object orientation, as provided by UML, proves useful to guarantee a good level of reusability, flexibility and also specification of needed interfaces. However, the object paradigm alone is not sufficient in our opinion and according to the WIRES experience, to ensure that all the aspects typical of a modern application are correctly taken into account in a development methodology, where also technological and operational aspects play a decisive role since the early development stages. Recent attempts are in the direction of providing business modelling capabilities to UML using business patterns [47] in the line of reusability.

Hence, a richer set of concepts, like those provided systematically in WIRES, should be applied in order to develop interoperable services successfully. Moreover, a suite of tools is strongly needed to support WIRES. We have started to develop tools for selection of candidate components from the repository using similarity criteria. Results on the tool support aspect are reported elsewhere [18,45] and in Damiani et al. [8], where we illustrate the tools developed in WIDE and in the AIPA Project (a project where we analysed schemas for the Italian public administrations). Of course, UML standard development tools can be employed to produce the various UML diagrams and to compile development documentation;
however, more advanced tools are needed to support business analysis and mapping from analysis to design. Instead, in our opinion, mapping rules provided by WIRES for implementation claim both for human decision about the best implementation choice and strategy and automated tools for other implementation activities, such as mapping activity diagrams into workflow languages or tools for database design.

Finally, we briefly comment on a topic mentioned in the paper and currently among our research issues. The aspect of dynamic invocation of services is concerned with the possibility of describing in an executable language the needed component services of a workflow application. We consider that implemented (i.e., WfMS executable) workflow components can be stored at different sites of a distributed system on various platforms; the workflow application specifies only a set of required functionalities to be fulfilled by the global workflow. Then, the WfMS is autonomously able to activate the needed workflow components at the right sites. We are currently investigating on the design of a Trader module, which, in the line of previous proposals [36,48], stores the description of the steps to be executed to complete a given service. The task of the Trader is to coordinate the local executions of the application portions needed at the various steps of the service. For this aspect, we are designing a proper set of language constructs and a tools architecture for the Trader.

Finally, future work regards the evaluation and operational test phases of WIRES in real e-service development. The overall goal is to obtain data about the technical and organisational validity of the approach. Experiments are being conducted for e-business applications using test scenarios in an industrial environment and some of the standard commercial WfMSs for which test routines are being developed. We expect feedback regarding the correct operations of exceptions, of distribution aspects, of suitability for real organisation environments, and the timeliness of the development of e-services. We intend to evaluate WIRES also for design and management of e-services.

Acknowledgements. The authors are thankful to the partners in the WIDE Consortium, to Silvana Castano and Luciano Baresi for work on reusable components and on preparatory methodology considerations, and to Ernesto Damiani for work on selection techniques from reusable components repositories. This work has been partially funded by the WIDE Esprit Project no. 20280, by Hurst 40% Projects Interdata and Data-X, and by MURST 5% Project VISPO.

References

4. Wieger O. Business process modeling and workflow definition with UML. SAP AG presentation, 1998
43. Ceri S, Pernici B, Wiederhold G. Distributed database design methodologies. IEEE special issue on distributed database systems, May 1987