Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects

— Habilitationsschrift —

vorgelegt an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

von
Dr. rer. nat
Mathias Weske

Münster
2. Dezember 1999

Preface and Acknowledgments

This thesis was written while I was a research assistant in the database and information systems research group of Prof. Dr. Gottfried Vossen at the Institute for Information Systems, Westfälische Wilhelms-Universität Münster.

First of all I like to thank Gottfried Vossen for his support of the research reported on in this thesis. In addition, he has been actively involved in the work on WASA from the beginning, and many papers in the early phases of the project were co-authored with him.

WASA started as an international cooperation project in 1994, in which the University of Campinas, Brazil, and the University of Münster participated. Valuable insights in the specific properties of scientific applications as well as a generic, workflow-based architecture were developed with our Brazilian partners Prof. Dr. Claudia Bauzer Medeiros and Prof. Dr. João Meidanis. Application studies in the context of workflow management in scientific applications were conducted with our Brazilian partners in molecular biology and geo-processing and with Torsten Reuß in laboratory information management. Their work is gratefully acknowledged. WASA is a team effort. I like to thank Gregor Wittkowski and Bernd Focke for their work on implementing the first prototype. Dominik Kuropka, Thomas Serries, Jens Hündling, and Hilmar Schuschel were involved in the development of the second prototype. Many thanks for their interest and involvement in WASA and the valuable work they have put into the system.

I like to thank the people to whom we demonstrated the system for their remarks and comments, a considerable number of which have led to improving the system. Among the persons who were patiently enduring WASA demonstrations and asked fruitful questions were Prof. Dr. Joachim Biskup, Prof. Dr. David Embley, Prof. Dr. Johann Christoph Freytag, Dr. Gerd Hillebrand, Dr. C. Mohan, Prof. Dr. Hans-Jörg Schek, Prof. Dr. Dennis Shasha, Dr. Dan Suciu, and Prof. Dr. Krishnamurthy Vidyasankar.

The work on workflow application development processes is based on joint work with Thomas Goesmann, Dr. Roland Holten, and Dr. Rüdiger Striemer. Thanks for the interesting discussions we had and the results we obtained in, let's say, casual environments.

Last and foremost I thank Daniela and Jonathan for their patience and support during the months of writing this thesis.

Münster, December 1999 M.W.

Summary

This thesis deals with the formal foundation, conceptual design, and prototypical implementation of workflow management systems. Based on properties of information systems applications, research issues in workflow management and requirements of suitable workflow support are characterized. In particular, the need for flexibility and dynamic adaptations is identified, i.e., the ability of workflow management systems to adapt running workflow instances to changes in the environment of application processes. Re-use of workflow schemata, the integration of application objects as well as high scalability and availability are characterized as additional important properties of workflow management systems.

As a formal foundation, a workflow language based on directed graphs is introduced, using a mathematical formalism. In this formalism, workflow schemas and workflow instances are described. Specific properties of workflow schemas like the prefix-closed property with respect to workflow schema consistency are formalized. Workflow application development processes are characterized as specific software processes, and a workflow design methodology is proposed.

The conceptual design of workflow management systems is studied in some detail. After a brief characterization of the conceptual design and the architecture of a workflow management system, the potentials for improvements are identified. Goals for the conceptual design and prototypical implementation of a novel workflow management system are derived. Since a mathematical formalization is not sufficient as a basis for the conceptual design of a workflow management system, object-oriented design methods are used. Workflow schemas, workflow instances, and related artifacts are modeled as objects. Since the resulting class diagram specifies the structure of workflow-relevant objects, it amounts to a workflow meta schema. The workflow meta schema is designed to support dynamic adaptations of running workflow instances by changing the relationship of a workflow instance object to a workflow schema object. To control dynamic adaptations properly, correctness criteria using the mathematical formalization are introduced.

Since workflow applications are typically performed in heterogeneous and distributed environments, a distributed object-oriented middleware is used as an implementation platform. Due to the object-oriented approach on the conceptual level, the system design and the implementation platform match nicely. A set of fundamental services are designed and implemented. The usage of the system is illustrated by a sample workflow application, which also introduces a workflow client application. The workflow client application can be configured to the functional requirements of different groups of persons involved in workflow applications.

Zusammenfassung

Diese Habilitationsschrift beschäftigt sich mit den formalen Grundlagen, dem konzeptionellen Design und der prototypischen Implementierung von Workflow-Management-Systemen. Es werden wissenschaftliche Fragestellungen identifiziert und Anforderungen an Workflow-Management-Systeme formuliert, die sich aus den Anforderungen moderner Anwendungen ableiten. Flexibilität wird als eine der zentralen Eigenschaften formuliert, insbesondere die dynamische Adaptierbarkeit laufender Workflow-Instanzen an neue Workflow-Schemata. Wiederverwendung von Workflow-Schemata und Anwendungsobjekten sowie hohe Skalierbarkeit und Verfügbarkeit werden als weitere zentrale Anforderungen an Workflow-Management-Systeme formuliert.

Als formale Grundlage für das konzeptionelle Design von Workflow-Management-Systemen wird eine auf gerichteten Graphen basierende Workflow-Sprache eingeführt und mathematisch formal beschrieben. In dem entwickelten Formalismus können spezielle Eigenschaften von Workflow-Schemata, etwa die Präfix-Abgeschlossenheit von Workflow-Schemata bezüglich Konsistenzeigenschaften, gezeigt werden. Die Eigenschaften von Prozessen zur Entwicklung von Workflow-Anwendungen als spezielle Software-Entwicklungsprozesse werden diskutiert, und eine Methodologie zur Entwicklung von Workflow-Anwendungen wird entwickelt.

Der konzeptionelle Entwurf von Workflow-Management-Systemen wird ausführlich untersucht. Nach einer kurzen Charakterisierung von Entwurf und Architektur eines Workflow-Management-Systems werden Verbesserungsvorschläge formuliert. Ziele für den konzeptionellen Entwurf und die protoypische Implementierung eines neuen Workflow-Management-Systems werden hergeleitet. Weil die mathematische Formalisierung für den konzeptionellen Entwurf nicht hinreichend ist, werden objekt-orientierte Methoden verwendet. Workflow-Schemata und Workflow-Instanzen werden als Objekte modelliert. Das resultierende Klassendiagramm beschreibt die Struktur workflow-relevanter Objekte und stellt daher ein Workflow-Meta-Schema dar. Das Workflow-Meta-Schema erlaubt dynamische Adaptionen laufender Workflow-Instanzen dadurch, dass Beziehungen zwischen Workflow-Schema- und Workflow-Instanzobjekten zur Laufzeit verändert werden können. Korrektheitseigenschaften für dynamische Adaptionen werden auf der Basis des mathematischen Formalismus definiert. Basierend auf gültigen Abbildungen (valid mappings) kann das Workflow-Management-System entscheiden, ob eine Workflow-Instanz an ein gegebenes Workflow-Schema angepasst werden kann.

Weil Workflow-Anwendungen typischerweise in verteilten, heterogenen Umgebungen ausgeführt werden, wird als Implementierungsplattform eine verteilte, objekt-orientierte Middleware eingesetzt. Konzeptioneller Entwurf und Implementierungsplattform passen somit gut zueinander. Basisdienste werden entworfen und implementiert. Die Verwendung des Systems wird durch einen Beispiel-Workflow illustriert. Dabei wird auch eine Workflow-Client-Anwendung als Benutzerschnittstelle des Systems vorgestellt. Diese Anwendung kann unter Beachtung der funktionalen Anforderungen unterschiedlicher Benutzergruppen konfiguriert werden.

Contents

Part I

Formal Foundations

Part I provides the foundations of this thesis. It motivates workflow management from an application-oriented point of view, it derives research issues in workflow management, and it sketches the requirements for suitable workflow management support from a technological perspective. Basic notations on workflow management are introduced, and the design of a workflow language based on directed graphs is presented formally. The language introduced will be used in the remainder of this thesis to specify workflows. A chapter on workflow application development processes proposes a workflow development methodology aiming at improving the planning and conduction of workflow projects.

Chapter 1

Introduction

Workflow management aims at modeling and controlling the execution of application processes in dynamic and heterogeneous organizational and technical environments [67, 32]. Hence, the motivation for workflow technology comes from the specific requirements of information systems applications. To provide a concise motivation for workflow management in general and for the topics addressed in this thesis in particular, specific properties of information systems applications are introduced. Based on these properties, important research issues in workflow management are derived, and a formal foundation of workflow management is introduced, which serves as a basis for solving these issues. A key aspect in computer science is developing theoretical results and showing their validity in operational prototypical software systems. Based on theoretical results with respect to the research issues addressed, this thesis discusses the conceptual design of workflow management systems. The concepts and methods introduced are validated in prototypical implementations of workflow management systems.

Background, Motivation, and Research Issues

The last decade of the 20^{th} century has brought severe changes to how organizations in commerce, public administration, education, and in science and engineering operate, how they interact with other organizations and with their clients. Many of these new developments are fueled by recent technological advances, for example the availability at low cost of computing and network resources and, most prominently, the rapid growth of the Internet. As a result, individuals can now communicate conveniently with colleagues and friends in other parts of the world, characterized by the global village metaphor. In the business sector, technological advances have created what is now seen as the global economy.

Probably the most obvious appearance of these developments is the great interest in online applications, i.e., attracting potential clients on the Internet, accepting orders, exchanging product specifications or scientific data and, to some extent, providing online delivery of goods and facilitating negotiation and payment via the Internet. In these settings, information, products, and services are now available to clients world wide instantaneously after they are released. This property of online applications have strong implications on their design, implementation, and maintenance of information systems supporting these applications. In particular, it motivates the need for fast creation of new services and the flexible adaptation of existing ones. On a technical level, system failures are no longer acceptable since they are immediately exposed to clients, who are likely to choose another organization to provide the desired services. As will be motivated in the remainder of this chapter, these requirements impose interesting research issues in workflow management, which can be solved by using a variety of computer science concepts, methods, and techniques. Before discussing the research issues addressed in this thesis and the concepts, methods, and techniques used for solving them, some general remarks on the application background of workflow management systems are appropriate.

From an application-oriented point of view, an important approach to cope with the new requirements is the explicit modeling of different aspects of applications and of the environment in which they are executed. The most prominent individual aspect in this context is process orientation, which has become a widely accepted and a successful approach to describe how work is performed in organizations. In the process orientation approach, application processes are in the center of attention rather than individual functions performed during these processes. To motivate process orientation, a brief look at the traditional approach to model work in organizations is taken. In the traditional approach, work is decomposed into fine granules, called functions. Work is organized around functions, and the relationship between functions is typically not taken into account. Despite of the fact that functions are performed properly by the individual applications, work in these settings is often performed inefficiently. Reasons for this fact include a variety of media to store application-specific data as well as redundant work and waiting times introduced by poor management of the interrelationships between individual functions.

As was realized in manufacturing earlier where production lines have been introduced decades ago to enhance the productivity of manufacturing goods, work is typically performed in processes. This means that the relationships between individual activities have to be taken into account, and work should be organized around processes rather than around individual functions. This observation also holds for processes which manage information rather than physical goods; these processes are called information processes.

Information processes appear in the domains mentioned above. In commercial organizations, insurance claim management and credit request processing are typical examples of application processes. A sample application process in public administrations is processing a proposal to construct a private house. Student enrollment in a university involves a number of steps, which can be characterized as an application process in the education sector. Laboratory information management, experiment management, and the construction of complex apparatus are typical examples of application processes in science and engineering.

The explicit modeling of processes first appeared in them context of applications in commercial organizations; application processes in these settings are called business processes. The notion of a business process is defined by Hammer and Champy [42] as "a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer". Hence, business processes typically involve mission-critical processes of an organization. A good indication of mission-criticality is that the organization was founded to perform these processes [68]. The activity of modeling business processes is called business process modeling. It is worth noting that business process modeling is not restricted to application processes in commercial organizations. In contrast, it can also be applied to other areas in which application processes need to be modeled, for instance in public administrations, education, and in science and engineering. Abstracting from the particular domain, the terms application process and business process can be used interchangingly. The design of business process models is typically developed by business domain experts rather than by computer scientists. Issues related to business process modeling will therefore not be addressed in this thesis.

After an organization has captured its business processes, software development projects are faced with the decision on how to support these processes properly. At this point workflow technology comes into the picture. As indicated above, workflow management aims at controlling the execution of the automated parts of business processes. Hence, it has to offer formalisms to express the automated parts of business processes and the technical and organizational environment in which they are executed. In general, these formalisms are known as workflow languages. Using workflow languages, workflow specifications are created, which are used as inputs for workflow management systems. Therefore, workflow languages have to be formal, so that the workflow management system can unambiguously control workflows according to given workflow specifications.

Based on the specific requirements of information systems applications, an analysis of the conceptual work as reported in the literature, and the limitations of existing workflow management systems, research areas in workflow management and corresponding functionality of workflow management systems are described:

- *Flexibility and Dynamic Adaptations:* While flexibility has been one of the major motivations for workflow technology from the beginning, only limited functionality with respect to flexible workflow management is provided by traditional workflow management systems. Today, the need for additional flexibility is widely accepted in the workflow community, indicated by a number of recent workshops [101, 61, 19].

- *Integration, Re-Use, and Software Components:* The re-use of workflow specifications to assemble new workflow specifications from existing ones is an important aspect in workflow management. A formal foundation of workflow specifications is an important prerequisite in this context. On a more technical level, the integration of external applications and existing software components is a promising approach.

- *Scalability, Availability, and Distributed Workflows:* Scalability and availability are important aspects which have to be satisfied by information systems, indicating the ability to enhance the throughput by installing additional resources, and providing services without failures, respectively. Distributed workflow management is a promising approach to enhancing scalability and availability of workflow management systems, as will be discussed in detail below.

In the remainder of this section, these research areas is motivated in some detail, research issues are identified, and conceptual and technical requirements of suitable workflow support are derived.

Flexibility and Dynamic Adaptations

Given the dynamic structure of today's organizations in commerce, public administration, education, and in science and engineering, it is unlikely that application processes are modeled once to be executed repeatedly without any changes. On the contrary, processes may change gradually over time to reflect changes in the environment of the process — processes evolve. Of course, flexibility is not restricted to software systems. According to the dictionary, an artifact is flexible if it can "easily be changed to suit new conditions". Flexibility in workflow management is used with this meaning: A workflow application is flexible if it can easily be changed to suit new conditions, imposed by changes in the environment of the application process. Depending on the application domain under consideration, there may be quickly evolving application processes and there may be slowly evolving ones. In some application domains there may even be processes which do not change at all, for instance where process structures are defined by legal regulations, which

are not expected to change over long periods of time. For these application processes traditional workflow support will suffice, for others, it won't.

As indicated above, flexibility has been a major motivation of workflow management from the beginning: By explicitly modeling processes and a variety of additional aspects to be discussed later in this thesis, individual dimensions in workflow specifications can be subject to change without changing other aspects. As will be indicated below, workflow languages are designed in a modular way that allows to perform changes to certain aspects, while preserving other aspects.

In general, workflow management systems provide the flexibility to perform changes to workflow specifications, and all future workflows will use the new, improved workflow specification. However, workflow specifications for given workflows cannot be changed in traditional workflow management systems. This means that once a workflow has started, no changes can be applied to its structure. This is a severe limitation of workflow management systems, since workflows can be long-running activities, which may run for days, weeks, or even for months or years. Restricting modifications to new workflows would render workflow technology virtually unusable in these settings, since long-running workflows would have to continue with a superseded workflow specification for long periods of time. It would be much more adequate to allow running workflows to be adapted to new and improved workflow specifications. Hence, the requirement to react quickly to changes in the environment of a process is not matched by the functionality of traditional workflow management systems. Most prominently, dynamic adaptations of workflows to new workflow specifications is regarded as an important functionality to enhance flexibility of workflow management systems.

The remainder of this thesis introduces the formal foundation, conceptual design, prototypical implementation, and the usage of dynamic adaptations of running workflow instances. It is not surprising that dynamic adaptations impose considerable requirements on the design of workflow languages as well as on the theoretical basis, conceptual design, and implementation of workflow management systems. By using correctness criteria based on formal specifications of workflows, dynamic adaptations can be controlled properly. Today, flexibility issues in general and dynamic adaptations of running workflow instances in particular is an important research topic in workflow management, and the results presented in this thesis will contribute to this research area.

Integration, Re-Use, and Software Components

Organizations typically operate in complex technological and organizational environments. In the technical environment, for instance, organizations may rely on a variety

of different information systems to store, retrieve, manipulate, and manage application-specific data. Some of these systems may use relational database technology, while others may use pre-relational systems. As a typical pattern, the technical and organizational environment of a company has gradually evolved since the company was founded. Often the mission-critical application systems of a company are the oldest and most reliable ones, because the company was founded to perform these operations. These application systems are known as legacy systems. Due to the high risk and considerable resources involved, re-implementing the software systems at the core of a business is a task often avoided. Not surprisingly, these applications have typically been developed without process orientation or workflow management in mind. In particular, they often lack the flexibility which is required to adapt applications quickly and with little effort to new requirements as they emerge. Workflow management systems are used to provide this functionality. Therefore, workflow management systems have to communicate with these applications to support mission-critical business processes.

The integration of legacy systems in workflow applications is one of the most important issues in workflow management from a practical perspective. Its importance comes from the mission-criticality of legacy applications. As a result, not only the legacy applications have to provide high performance and availability, but also their integration in the workflow application. Approaches to integrate legacy applications include wrapping techniques, in which the legacy system is wrapped with a software layer, called wrapper. The wrapper hides the particularities of the legacy system and presents a clean interface to the workflow management system. This interface is used to invoke legacy applications, to transfer input data to and to get output data from these systems.

Another requirement of business applications is the fast development of new services or the fast improvement of existing ones. The approach of creating new services by assembling processes or process fragments leads to faster design and implementation of these services. In this context, the presentation of services to customers using Internet technology is important, since new services can be made available instantaneously. The fast and convenient creation of new application processes and their implementation using workflow technology requires support on two areas: First, re-use of existing workflow specifications in the context of multiple applications. Second, the convenient integration of application programs in workflow applications. Issues related to these areas are discussed in turn.

In order to create new workflows from existing ones, workflow specifications have to be re-used in different workflow specifications. A formal specification of workflow schemas and how workflow schemas can be assembled is a prerequisite to support workflow schema re-use. A workflow specification library can be build as a repository for

application processes and their modeling and implementation as workflows. A workflow specification library can also be used to create new workflows from fragments of existing ones, as discussed above. This approach is closely related to component-based software development: Workflow specifications are regarded as software components, and by assembling workflow components, new workflow specifications can be designed conveniently. The introduction of new services by creating new application processes cannot be achieved without adequate technological support.

In this context, application frameworks are a promising approach. These frameworks aim at efficient development of complex software system by providing basic functionality and defined ways to extend it. Instead of starting from scratch, the system development starts from a set of predefined classes, which are then extended to suit the needs of a particular application. Frameworks are typically based on object technology, since abstraction concepts provided by the object paradigm — for instance, inheritance, aggregation and polymorphism — prove useful in this context. This approach aims at developing objects and collections of objects which have a meaning in the application domain. These objects are known as business objects. In a Request For Proposals for a Business Object Facility [77] issued by the Object Management Group (OMG), business objects are defined as follows:

> A business object is defined as a representation of a thing active in the business domain, including at least its business name and definition, attributes, behavior, relationships, rules, policies and constraints. A business object may represent, for example, a person, place, event, business process or concept. Typical examples of business objects are: employee, product, invoice and payment. The business object [...] is implemented by one or more objects in the information system.

Business objects and business-object frameworks are also useful with respect to the integration of domain-specific applications, using wrapper-techniques. In addition, software components in general and business objects in particular are useful for fast and efficient development of new application systems or improvement of existing ones [104]. Since in contemporary businesses, new services can often be implemented by new applications or new business processes, component-based software development is a promising approach.

Business objects can be re-used as software components in a variety of applications. Hence, software development and maintenance can be simplified considerably using business objects. The advantages of software components can be exploited even more when they are integrated by a workflow management system, which adds process support to

business objects. To go one step beyond, workflow functionality can also be designed and implemented by specific business objects. This approach integrates the application-specific functionality and workflow functionality in a common framework. In this thesis, an object-oriented approach to the conceptual design and the implementation of workflow management systems is proposed, which simplifies the integration of workflow functionality, software components, and application frameworks. Concepts, architectures, and upcoming standards in the context of business objects and business-object frameworks are discussed in [112].

Availability, Scalability, and Distributed Workflows

High availability is an important aspect of workflow applications, often characterized by 24/7 availability, meaning that the applications have to be operational 24 hours a day, 7 days a week. This means that down times due to system administration activities are also not acceptable. In the workflow context, installing new or modified workflow specifications or changing the organizational and technical environment of the workflow application have to be made while the system is operational. Safe operation and fault tolerance are related features in this context. This means that in the event of a system failure, after the system restarts, it has to be recovered to a consistent state. In the database context, similar requirements have led to the transaction concept, involving synchronization of parallel transactions and automatic recovery after system failures [7, 37]. In particular, after a system failure, the database will be restored to a consistent state using restart procedures. In the workflow context, at each point in time the current status of the workflow management system has to be stored persistently. Persistently stored workflows are therefore a prerequisite to recover the current state of the workflow management system after a system failure has occurred.

Recently there is an increasing interest in application processes involving multiple organizations. In particular, workflow management is seen as an important enabling technology to facilitate workflows between multiple enterprises. This emerging field of workflow management is characterized by the term interorganizational workflow. From an technological perspective it is not reasonable to control interorganizational workflows in a single dedicated machine. This would lead to a situation in which the internal workflows of one organization are controlled by a workflow server run by another organization. Most likely, acceptance issues would occur in this context which would make this approach not feasible. Distributed workflow executions can be motivated by interorganizational workflows, in which all participating organizations have the opportunity to control the parts of the overall workflow, for which they are responsible. Due to the fine granularity of work-

flow execution control, group-wide workflows can be controlled on a local machine, while department-wide workflows may be executed on a dedicated workflow server running under the control of that department. Finally, interorganizational workflows can be implemented conveniently by placing the relevant parts of the global workflow to dedicated machines of the organizations involved. Hence, local autonomy, which is an important issue from an organizational point of view, is supported by the software infrastructure.

Since workflow management aims at supporting mission-critical application processes, a large number of concurrently executed workflows can be expected. Hence, the workflow management system has to cope with a heavy load, which is characterized by a large number of workflow activities and many persons involved, i.e., the workflow management system has to provide scalability, meaning that the system can perform workflows faster or more workflows can be performed if additional resources are installed. As a consequence, workflow applications have to be scalable. Adding resources to a system which uses a fully distributed approach improves the efficiency and enhances the throughput, provided additional communication cost incurred by additional resources does not hamper these advantages.

Workflow Application Development Processes

Workflow applications are developed in workflow application development processes, as specific software development processes, in which typically numerous persons with different backgrounds and experiences are involved. Software development processes are studied in software engineering, and workflow development processes are identified in this thesis as software development processes with a number of specific properties. To cope with the properties of workflow development processes, these processes are studied, and a workflow development methodology is introduced, aiming at assisting workflow project participants in planning and conducting workflow application development projects.

Research Issues not Addressed

While this thesis addresses a number of important research issues in workflow management, there are other important research issues which are not addressed; some important research directions in workflow management are sketched in the remainder of this section. We remark that this section does not discuss related work (an extensive analysis of related work will be given in Part III). In contrast, research topics are briefly sketched, and references to important contributions addressing the respective topics are provided. The reader may consult these references for more information.

One area of workflow management research deals with architectures of workflow management systems, based on different goals. One goal is the development of workflow servers according to particular aspects of workflows [56]. Other approaches in workflow management architectures consider partitionings of workflow servers in compliance with the organizational and technical structures of an organization [117, 116] as well as typical access patterns to reduce communication overhead, using role information [4, 85]. In [34], the use of queuing theory for the partitioning of workflow servers is discussed. The integration of workflow technology and computer supported cooperative work is an important research issue in the workflow context [59]. By integrating methods and tools for supporting collaborative work with methods and tools for modeling and controlling the execution of activities, advanced information systems can be developed. Taking advantage of a strong background in database systems, a number of research projects deal with transactional workflows and transactional processes [93, 99, 89]. Approaches in this context aim at defining execution guarantees for workflows, just like execution guarantees for application programs in the context of database transaction processing. Active databases are another background for research in workflow management. The application of methods and techniques from the area of active databases are twofold. First, workflows can be modeled using so called event-condition-action rules, a basic formalism in active databases. Second, active databases can be used as implementation platforms for workflow management systems [33]. The combination of workflow technology and product data management is an interesting approach for supporting complex application processes in engineering with workflow functionality [70].

Only recently, another application area of workflow management came up: electronic payment. In this context, advanced security and authentication mechanisms are used to make sure payment is done in a reliable and safe manner; workflow management systems can be used for modeling and controlling the steps that occur during an electronic payment [2, 94]. Temporal aspects are another research issue in workflow management. The main idea of work in this area is modeling the duration of workflow activities explicitly; there are time intervals for each workflow activity, during which it has to be executed [25]. Finally, migrating workflows is mentioned as a research issue in workflow management. The basic idea in migrating workflows is that workflows can migrate to particular sites of a given organization or even to sites maintained by other organizations [90], with the aim to make use of local knowledge and expertise, and as a means to provide load balancing. Approaches in the context of migrating workflows are related to agent technology, in which autonomous agents with defined goals get in contact with other agents in order to negotiate and, finally, to reach their goals.

General Approach, Positioning of Work, and Overview

This thesis investigates research issues in workflow management within the computer science domain. In particular, the formal foundation, conceptual design, and prototypical implementation of workflow management systems is addressed. Hence, this thesis resembles rather closely Aho and Ullman's understanding of computer science, who in [1] state that

> [. . .] fundamentally, computer science is a science of abstraction — creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.

In order to succeed in this challenge, a variety of abstraction concepts, methods, and techniques known from computer science are used. For instance, for a suitable abstraction from real world application processes, workflow specifications have to be developed, whose structure is defined by workflow meta schemas. A mathematical formalization of workflow meta schemas is required for correctness considerations as far as the execution of workflows and the dynamic adaptation of running workflows is concerned. The workflow meta schema introduced in this thesis is based on a variant of directed graphs. Graphs are well-known structures in computer science, which are used in a variety of contexts, ranging from the development of compilers to chip layout and transaction processing in database systems. One of the nice properties of graphs is that they combine graphical representations of artifacts with a thorough mathematical foundation.

However, mathematical abstractions in general are not adequate for the conceptual design of complex software systems like, for instance, workflow management systems. Hence, additional abstraction mechanisms have to be used. For a number of reasons (to be discussed later), object-oriented modeling techniques are used for the conceptual design of complex software systems in general and workflow management systems in particular. As opposed to the mathematical formalization, which describes the domain in terms of products, functions, mappings, and the like, the object-oriented model provides abstraction concepts like inheritance, aggregation and polymorphism, which are well established concepts to describe the conceptual design of complex software systems. Since workflows are dynamic objects in general, their behavior has to be described properly. In order to do so, state transition diagrams as a specific form of finite automatons are used. That formalism allows to explicitly maintain information on the states of workflow objects, which is used to make sure state transitions of a workflow can only occur in compliance with their behavioral specification.

Database systems are widely used to store, manage, and access efficiently large volumes of data in a variety of application domains. Since workflow management systems

deal with mission-critical business processes, database technology is adequate to store workflow-related data persistently. In order to do so, techniques from conceptual database design are used to develop a database schema suitable to store and access efficiently workflow specifications. Once the database schema is specified and the database is installed, it can be populated with workflow specifications. On a higher level of abstraction, persistent storage of workflow objects and application objects can be made available by software systems which reside in between application systems on the one hand and fundamental services on the other hand. Due to their relative position, these software systems are called middleware. In object-oriented middleware, persistency can be provided as a service to all objects of a given computation environment. This approach allows to transparently store arbitrarily structured objects in persistent storage. The usage of services in distributed object-oriented middleware can be specified by interface definitions. As determined by the object paradigm, the interface of an object is separate from its implementation.

Workflow management is an attractive research domain in computer science, since methods, concepts and techniques from a variety of computer science disciplines can be utilized. To position the work presented in this thesis both horizontally, i.e., in terms of the computer science disciplines involved, and vertically, i.e., in the levels of abstraction considered, the previous paragraphs can be summarized as follows: To characterize the horizontal breadth of the work, abstraction concepts, methods, and techniques from the areas of graph theory and mathematical formalization, object-oriented design and behavioral modeling, conceptual design of database schemas, and object oriented middleware as well as software development processes are involved. In terms of the vertical positioning of the work, the topics addressed range from high-level issues like the analysis of the requirements of advanced business applications, to mid-level activities like the conceptual design of workflow management systems, to low-level topics like the conceptual design and prototypical implementation of workflow management systems and of particular services based on distributed object technology.

As a demarcation of this thesis from the business computing domain, the following remarks are appropriate: Business computing deals with applying or developing methods and techniques to solve specific problems or to enhance the efficiency of particular businesses. For example, formalizing application processes by business process models and improving them are typical activities of business computing experts. The work of business computing experts in general considers application-specific issues. As far as software support is concerned, from the software systems available in the market the ones that match best the particular requirements of the business under consideration are selected, customized, and used to solve particular business issues. Business computing does not

primarily deal with the conceptual design and implementation of software systems. For example, to realize flexibility in workflow management systems, methods and techniques from business computing are not adequate. Instead, computer science concepts, methods and techniques are suitable. Since this thesis utilizes computer science concepts, methods, and techniques to solve research issues related to the formal foundation, conceptual design, and prototypical implementation of workflow management systems as a specific form of complex software systems, this thesis is positioned in the computer science domain.

The remainder of this thesis is organized as follows: Part I provides the formal foundations of the thesis. In Chapter 2, basic workflow terminology is introduced, and workflow aspects are described to classify dimensions which are relevant for workflow management and which consequently have to be taken into account when designing workflow languages. A workflow language based on directed graphs is described formally in Chapter 3. In particular, a formal specification of workflow schemas and workflow instances is proposed, which is based on a mathematical formalism. In addition, it can be used as a basis for developing the conceptual design of workflow management systems supporting that workflow language, especially as far as correctness issues of flexibility operations are concerned. Workflow application development processes are investigated in Chapter 4. A workflow application development methodology is proposed. While the methodology does not automatically create successful workflow projects, it does help project managers and project participants in planning, organizing and conducting workflow projects.

Part II reports on realization concepts for workflow management systems. Chapter 5 sketches the results of early project phases. Experiences gathered in molecular biology, geo-processing, and laboratory information management showed a strong need for process support to model complex application processes and to control the execution of these processes. These observations led to the definition of a generic architecture, the WASA architecture. (WASA is an acronym for Workflow-based architecture to support Scientific Applications.) To show the validity of the conceptual work, a prototype based on database and Internet technology was designed and implemented. Based on lessons learned during the initial project phases, Chapter 6 introduces the conceptual design of a novel workflow management system. In particular, a workflow meta schema describes both the components of a workflow and their relationships from a logical perspective. State transition diagrams are used to describe the dynamic behavior of workflows. Chapter 7 discusses dynamic adaptations of running workflow instances to new workflow specifications, focusing on conceptual issues and correctness considerations as well as on realization strategies for dynamic adaptations. In Chapter 8 an implementation of a prototype realizing the

theoretical concepts is presented. The implementation is based on a distributed, object-oriented middleware, which is used both as a communication platform and as an implementation platform using a number of fundamental services. A configurable workflow client application and a sample workflow application are described in Chapter 9, showing the usage of the system.

Part III discusses related developments and future work. Chapter 10 focuses on related work and system developments, concentrating on conceptual results and workflow management system development projects which are related to the work presented in this thesis. According to the aspects covered, related work in the areas of workflow language design and flexible workflow management, object-oriented and distributed approaches to workflow management system design, workflow management in scientific and engineering, and workflow application development processes is discussed. Finally, Chapter 11 presents conclusions as well as our plans for future work.

Chapter 2

Workflow Basics

This chapter introduces the foundations of workflow management in an informal way. Section 2.1 discusses workflow terminology and levels of abstraction in workflow management. The concepts introduced are illustrated with a sample workflow. Section 2.2 reviews dimensions which are essential to describe workflows; these dimensions are known as workflow aspects. Basic properties of workflow languages are introduced.

2.1 Workflow Terminology

While workflow management has been a research and development topic for several years now, there is still no terminology which is generally agreed upon. This fact is reflected by the presence of a number of homonyms and synonyms used in the workflow community. A term is a homonym if it is used with different semantic meanings; two terms are synonyms if they are used to describe the same semantic concept. An example of a homonym is the term workflow model. While some authors mean by workflow model entities and relationships from which workflow specifications are composed, i.e., descriptions of workflow specifications, others use workflow model to refer to a workflow specification itself. Synonymous terms in the workflow context are, for example, workflow activity and atomic workflow, which both characterize a unit of work as part of a workflow. These semantic ambiguities have to be resolved in order to present the concepts of this thesis in a clear and comprehensible way. The remainder of this section provides a coherent terminology used throughout this thesis.

2.1.1 Workflow Schemas and Workflow Instances

From a high-level point of view, workflow management deals with explicit modeling of the automated parts of application processes with the aim of controlling their execution in given technical and organizational environments. Workflow management systems are software systems which are designed to perform this task. In order to do so, a workflow management system needs a specification of the automated parts of application processes and the organizational and technical environment in which they will be executed. These specifications are known as workflow schemas. When an application process starts, an internal representation of the automated parts of the business process is created. This representation is called workflow instance.

The activities involved in a workflow instance, their structure and their organizational and technical execution environment is defined in a workflow schema. In general there are numerous workflow instances for a given workflow schema. For example, an order processing workflow schema may have hundreds or thousands of workflow instances, each of which represents the automated parts of a single real world application process. Workflow schemas define execution correctness criteria which have to be satisfied by the respective workflow instances. The main task of a workflow management system is to make sure the correctness criteria as specified in workflow schemas are actually satisfied by workflow instances.

Information systems applications which use a workflow management system to control application processes are called workflow applications. Typically, each workflow application runs numerous workflow instances concurrently, which can be based on multiple workflow schemas. It is assumed that workflow instances can be executed independently of each other. If there are close relationships between workflow instances then the workflow instances are probably better modeled as parts of a common, more complex workflow.

A variety of components are relevant for the execution of workflow instances, all of which have to be specified in a workflow schema. For instance the activities a workflow is composed of, their execution order, and the application programs used to perform these activities. The components that are modeled in workflow schemas are represented in workflow meta schemas. More precisely, a workflow meta schema characterizes not only the components of workflow schemas, but also their relationships.

To properly describe the artifacts involved in workflow applications, different levels of abstraction have to be identified. These levels are shown in Figure 2.1. In the highest level of the abstraction hierarchy, workflow meta schemas describe the components of workflow schemas and their relationships. Typical examples in this level are “activity”

	Workflow Management	Programming	Databases
Meta Level	Workflow Meta Schema	EBNF	ER Diagrams
Type Level	Workflow Schema	Program Code	Database Schema
Instance Level	Workflow Instance	Process	Database Values

Figure 2.1: Levels of Abstractions, Exemplified by Different Domains.

and "execution order" since virtually all workflow meta schemas support these concepts. In traditional software development, programming languages are specified in this level, often using the Extended Backus-Naur (EBNF) formalism for the specification of the syntactic structure of a programming language. In database technology, the meta level consists of languages to specify database schemas on a high, system-independent level of abstraction. Entity relationship diagrams are widely used for this purpose [16].

The second level of abstraction uses the entities defined in the meta level to specify valid entities on the instance level. This level can be thought of as the type level. In workflow management, workflow schemas are regarded as types, since they describe the structure and behavior of a set of workflow instances. On the other hand, workflow schemas are an instance of the workflow meta schema. An example of a workflow schema element is "CheckCredit", which specifies one specific type of workflow activity. In programming, the type level is represented by programs coded in programming languages. In database systems, database schemas appear in the type level.

In the instance level, workflow instances are internal representations of the automated parts of application processes; the structure of workflow instances is specified by workflow schemas. To this respect, there is an instance-of relationship between workflow instances and workflow schemas. Similarly, a process defined as a program in execution represents the instance level in traditional programming. Multiple invocations of a given program are possible, each of which amounts to a process, i.e., to a new entity in the instance level. In database systems, the values stored in the database represent the instance level. Just like the structure of workflow instances is specified in workflow schemas, the structure of databases is defined in the type level by database schemas.

Levels of abstraction similar to the ones shown in Figure 2.1 are also present in object modeling techniques, for instance in the Unified Modeling Language (UML) [79]. However, there is an additional layer on top, called the meta meta model layer, which defines languages for specifying meta models. In the UML context, for example, meta class and meta attribute sit in the top layer. The meta model is the next level of the architecture. Meta models are instances of the meta meta model; they define languages for defining

models. UML constructs in this level of abstraction are, for instance, class, attribute and association. In the model layer languages to define information domains are defined. Sample classes are customer, product, and order. In the bottom layer a specific information domain is defined, for instance customer “Petersen”, product “Tennis Racket” and order “OrdId123”.

To summarize, workflow schemas describe the automated parts of application processes. Workflow schemas are used by workflow management systems for the initiation and controlled execution of the automated parts of business processes. Once a specific application process is started, a respective workflow instance is initiated by the workflow management system. Workflow instances are internal representations of the automated parts of a business process. Different levels of abstraction can be identified to capture the complexity in modeling complex systems. Just like in database technology, programming languages, or object-oriented modeling, a meta level describes how types are specified. Types in turn describe the structure and behavior of instances. In the workflow context, a workflow meta schema describes how workflow schemas are composed. Workflow schemas determine the structure of workflow instances. Hence, entities in workflow can be classified according to levels of abstraction, just like in the other computer science disciplines mentioned.

2.1.2 Sample Application Process

In order to keep the presentation of workflow basics concise and to provide a common basis to present different workflow languages in the remainder of this thesis, an example of an application process from the area of credit processing in a banking environment is introduced. Using this example, the relationship between workflow meta schemas, workflow schemas, and workflow languages is discussed in more detail. The example originates from IBM’s workflow management system [52].

The application process is informally described as follows: The process starts when a customer requests a credit from a bank. The customer does so by filling in a credit request form and by submitting it to the appropriate department in the bank. The information in the credit request form is then transferred into the bank’s information system. After the validity of the data is checked, an assessment of the risk involved in granting the credit request is conducted. As governed by the policies of the bank, depending on the credit amount requested, checking activities of different complexity are required, and persons of different skills and competences are involved. We assume that the credit request checking activity is performed by a financial expert, who decides on the request based on the credit amount and the financial situation of the customer. If the domain expert in the bank

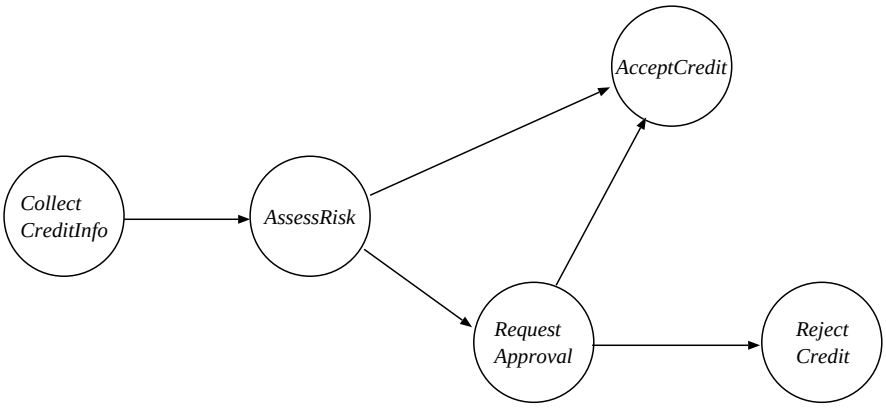

Figure 2.2: Simplified Workflow Schema.

decides to grant the credit, a set of administrational activities to allocate the credit amount and to transfer it to the customer's account are launched. If the credit request is not granted then a second, more advanced domain expert re-evaluates the case, possibly after getting hold of additional information on the financial situation of the customer. Depending on her judgment, the credit is rejected or granted eventually. While this description of an application process simplifies real world applications considerably, it does provide a basis for the presentation of workflow aspects involved and, below, of workflow languages.

Figure 2.2 shows a simplified workflow schema for the application process under discussion, using a directed graph. In this graph, workflows and their execution order are specified by nodes and directed edges between nodes, respectively. As a formal foundation, refinements of workflow schemas, workflow instances, and their formalization based on directed graphs is introduced in the next chapter.

2.1.3 Overview of Workflow Aspects

The approach to achieving a high degree of flexibility in modeling workflows is specifying different properties of the application process independently from each other, in a modular way. In order to change different parts of a workflow separately, the parts of workflow schemas are modularized by workflow aspects. Workflow aspects should be independent from each other, or orthogonal. Orthogonal workflow aspects allow to change certain aspects of a workflow schema without touching other aspects. To this end, workflow aspects modularize the specification of workflow schemas, with the aim of orthogonal specification of the parts a workflow schema is composed of. For instance, by defining separately an external application used to perform a specific workflow activity from the conditions under which that particular workflow activity is started then the ap-

plication program can be changed without changing the start condition of that workflow activity. Workflow aspects are discussed in some detail in the next section. However, the concept of workflow aspects are fundamental to workflow management, and basic workflow terminology can be introduced using workflow aspects. Therefore, workflow aspects are sketched at this point.

To describe the basic concepts of workflow aspects, the properties of workflows as specified in workflow schemas have to discussed in more detail. In general, the activities of a business process and, consequently, the activities involved in the respective workflow, are related to each other. Hence, workflow management typically deals with related activities, not with unrelated ones. Two well known forms of relationships between workflow activities are execution constraints and data constraints. Execution order is a specific form of execution constraint, which is supported by virtually any workflow language. If a workflow schema defines an execution order constraint between two workflow activities then the second activity can only start after the first workflow activity has terminated. For each workflow instance based on a given workflow schema, the execution order defined by the execution constraint of the workflow schema has to be met. It is the responsibility of workflow management systems to make sure all constraints defined in workflow schemas are satisfied by the respective workflow instances.

Data dependencies between workflow activities are the second form of constraints. Application processes are typically involved in generating, accessing, and manipulating application-specific data. In the credit request example, for instance, the customer data and data on the credit request are application-specific data. Workflow schemas capture data dependencies between workflow activities. Therefore, the structures of application-specific data have to be known to the workflow management system. A common pattern in application processes is the transfer of data between related workflow activities. In the banking example, the credit request data is generated by one activity, and it is transferred to follow-up activities, for instance to decide on granting the credit request. By modeling explicitly the flow of data between workflow activities, data dependency information is available and can be used to analyze, control, and monitor the flow of data through an application process represented by a workflow instance.

Application processes typically require existing software systems to perform certain workflow activities. A typical information system accessed during the execution of credit check workflows is the bank's accounting system. In general, it is not feasible to assume re-development of these applications when installing workflow technology in an organization. In contrast, the existing, reliable and proven software systems of an organization are used by the workflow to perform certain activities in a reliable and efficient way. These software systems are called legacy applications. The integration of legacy

applications is challenging from a technological point of view, since these systems have been typically developed without having workflow management or any other integrating software technology in mind. As it turns out, the integration of legacy applications is an important success factor of workflow applications, as will be discussed in more detail in Chapter 4. In terms of orthogonal workflow aspects, the change of an application program used to perform a workflow activity should not affect other aspects of the workflow, for instance the conditions under which it is being executed or the data dependencies defined between workflow activities of a given workflow instance.

The organizational integration of application processes is another dimension that has to be modeled in workflow schemas and which, consequently, has to be covered by workflow languages. To capture this aspect, workflow languages often provide a role concept. The role concept is an abstract concept, which describes predicates on the persons involved in a given organization. Roles describe skills, competences, responsibilities, and availabilities of these persons. Typically, each person can play one or more roles. Roles are assigned to workflow activities, and when the workflow executes, the workflow management system uses the role information to compute a set of persons to perform the workflow activity, a workflow management system functionality known as role resolution.

Workflow management applications involve different groups of persons, for instance workflow modelers, workflow management system administrators, workflow supervisors, and workflow participants. The task of workflow modelers is to use specifications of application processes to develop workflow schemas. Once the task of the workflow modeler is completed, a workflow management system administrator is responsible for installing the workflow schema in an operational workflow management system. Then workflows based on the installed workflow schemas can be initiated, executed, and monitored. Once a workflow instance is completed, it can be documented and analyzed. In addition, the workflow management system administrator is responsible for administering the system, for conducting general maintenance activities like user administration and providing data security. From a systems point of view, the work of a workflow management system administrator is similar to the work of a database system administrator in the database context. Workflow participants are domain experts, which use the workflow application to perform their work. In the banking example, the workers in the bank who access the workflow application are workflow participants. Typically, workflow participants use specific software components to access workflow applications. These components are provided by the workflow management system; they are known as workflow clients. Workflow supervisors are persons who are responsible from an application-specific point of view for the execution of particular workflows. For example, the manager of a depart-

ment is responsible for all workflows which are performed in the department she heads. We mention that the role concept can be used not only to model different roles workflow participants can play, but also the groups of persons administering and supervising the workflow application.

2.2 Workflow Aspects

After introducing the general idea of workflow aspects in the previous section, this section presents a set of common workflow aspects in some detail. These workflow aspects will be used to guide the presentation of workflow specifications in the next section. From a high-level point of view, workflow aspects can be characterized by simple questions:

- *Functional Aspect:* What is being done?
- *Behavioral Aspect:* When and under which conditions will it be done?
- *Information Aspect:* Which data will be used?
- *Organizational Aspect:* Who will perform it?
- *Operational Aspect:* Which software systems will be used to do it?
- *Flexibility Aspect:* What kind of changes can be applied to a workflow, and when?

Depending on the application domain addressed, different additional workflow aspects can be defined, for instance historical and causal aspects [56]. However, the workflow aspects characterized above suffice for the topics addressed in this thesis.

2.2.1 Functional Aspect

The functional aspect of workflow schemas describes what has to be done during the respective workflow instances. This aspect involves a textual description of the workflow activities which are necessary to perform a certain class of application processes. Since typically multiple sub-activities are required to perform an application process, the functional aspect also covers the functional decomposition of activities as present in application processes, i.e., it specifies which activities have to be executed within a workflow and what these activities are expected to achieve. To deal with the high complexity of application processes, the concept of nesting is often used to describe the functional decomposition of workflows. In particular, workflows are partitioned into complex and atomic workflows. Complex workflows are composed of a number of workflows, which

can be atomic or complex. Atomic workflows are also called workflow activities. From a workflow management point of view, an atomic workflow represents an atomic unit of work. Typically atomic workflows are performed by external applications with or without interaction by workflow participants.

The functional decomposition of the credit request workflow is specified by a set of activities which have to be carried out during a credit request application process. In the top level, the credit request can be modeled by a complex workflow schema, which is composed of a set of workflow schemas. Entering the data from the credit request form into the bank's software system can be modeled as an atomic workflow, which is implemented by an application program. Additional sub-activities of the process include activities to assess the risk of granting the credit request, to request credit approval, and to decide on the credit request, i.e., either to grant or to reject it.

It is important to recall that the functional aspect describes what has to be done during a particular workflow execution; it does not specify how it is done, i.e., which software systems are being used to perform particular workflow activities. In the credit check workflow, for instance, the functional aspect does not define how the data entering and checking is done — this is covered by the operational aspect, discussed below. Constraints on the execution of workflow activities are also not described in this aspect — these properties are defined in the behavioral aspect, discussed next.

2.2.2 Behavioral Aspect

To facilitate the controlled execution of workflow instances by workflow management systems, the behavior of sub-workflows of a given complex workflow have to be taken into account. In particular, the behavioral aspect specifies under which conditions the sub-workflows of a given complex workflow are executed. Important components of this aspect are control flow constraints and start conditions. Control flow represents the execution order of workflow activities. For instance, when in the application process a sub-workflow "assess risk" can only be started after sub-workflow "load credit request form" has terminated then a control flow constraint can be used to model this relationship. When the complex workflow starts, the workflow management system makes sure that the activities are executed in the order specified by the behavioral aspect in the workflow schema.

Depending on the workflow language used, there are additional concepts to define the behavioral aspect. Start conditions can be used to define a precondition for the execution of a particular workflow or workflow activity. Start conditions can be modeled as boolean functions, and a workflow is started only if its start condition is evaluated to true. Other

concepts to express the behavioral aspect are transition conditions between workflows or explicit branch and join nodes. These concepts will be discussed in more detail when sample workflow languages are investigated. The behavioral aspect is an integral part of workflow schemas; it is covered by virtually all workflow languages, and workflow management systems support mechanisms to guarantee that the execution constraints between workflows as defined in the behavioral aspect of workflow schemas are satisfied by workflow instances.

In the application process discussed above, for example, the behavioral aspect specifies that entering credit data is done before the checking for incorrect values is performed, which in turn is done before the risk is assessed and the decision on granting or rejecting the credit is taken. Branching of control flow is another concept of the behavioral aspect. In the sample application process, branching can be used as follows: If the amount is smaller than a predefined value x then a rather simple checking procedure is applied. If the requested amount exceeds x then a more complex procedure has to be performed to either grant or reject the credit request. Depending on the workflow language used, the semantics of branches can be parallel, alternative, or it can be controlled by predicates which are evaluated at execution time. An example of the latter form can be specified by $amount <= x$ and $amount > x$, respectively. It is worth noting that some workflow activities as specified in the workflow schema are not performed for each workflow instance based on that schema. Obviously, each complex workflow instance executes either the grant credit request or the reject credit request activities. Workflow languages provide means to define this kind of application semantics, for instance, using transition conditions or start conditions.

2.2.3 Information Aspect

An important aspect of workflow languages is modeling the structure of application data, since it allows workflow management systems to control the transfer of application data as generated or processed by workflow activities. In graph-based workflow languages, the information aspect is represented by data flow between workflow activities. In particular, each activity is assigned a set of input and a set of output parameters. On its start, an activity reads its input parameters, and on its termination it writes values it generated into its output parameters. These values can then be transferred to follow-up activities in the workflow, which use them as input data. This transfer of data between workflow activities is known as data flow.

By providing graphic language constructs to represent data flow between workflow activities, the information aspect can be visualized and used to validate and optimize ap-

plication processes. While the basic principle of the information aspect is straightforward, there are many technical issues to solve, for instance different data formats of the application program which generates the data and of the application program which uses the data. In this case, filters are required to allow seamless integration of different external applications using different data formats. To this end, it is desirable that data as specified in a data flow is strongly typed, rendering necessary a typing scheme for data which occurs as parameters of workflow activities. In doing so, potential typing incompatibilities can be detected during the workflow's modeling phase. In this context, business objects which encapsulate structure and behavior are an interesting approach, which is investigated in more detail below.

The information aspect in the sample workflow describes the data types involved, for instance, data types for customer data, credit forms, and risk assessments. Data flow constraints in the sample workflow occur between the activity in which the credit form is entered into the system and follow-up activities, which use this information to decide on granting or rejecting the credit request. In addition, there is a data flow from the decision taking activity to the activity in which the customer is informed of the result of his credit request.

2.2.4 Organizational Aspect

Workflows are executed in complex organizational and technical environments, and a major goal of workflow management is enhancing the efficiency of application processes by assigning work to persons or software systems as specified in workflow schemas. To reach this goal, a workflow management system has to be provided with information on the organizational and on the technical environment in which the workflows will be executed.

Since a strict assignment of workflow activities to persons is not feasible in most cases, the role concept is used. When a workflow activity is about to start, the workflow management system uses role information defined in the respective workflow schema to select one or more persons which are permitted, competent and available to perform the requested activity, a workflow management system functionality called role resolution. Depending on the requirements of the application domain and on the concepts supported by the workflow management system used, the role concept may have different levels of complexity. While some systems are restricted to a simple role concept, other systems may provide additional features for advanced role resolution, for instance, data dependent role resolution, delegation of workflow activities, and substitution of persons. Complex substitution rules may be defined, which even take into account the overall organizational

structure of the institution.

Data dependent role resolution describes the fact that role resolution uses data which will only be generated during the execution of the workflow. Hence, the data values are not known when the workflow schema is created. A popular example of this functionality is granting requests for leave of absence: The policies of an organization may govern that the department chairs grant the requests of all members of their respective departments. To avoid the department heads from deciding on their own request, the next level in the organizational hierarchy will be selected. In this case, role resolution is data dependent since the parameter "person" of the grant request workflow is used during role resolution. If the person is not the department head then the department head is selected, otherwise the manager of the department head is chosen to decide on the request for leave of absence.

People involved in the execution of the sample credit request workflow are members of the bank's credit department. Activities of the sample workflow are scheduled to persons in that department according to their positions, qualifications, and according to their availability, which are properties specified by roles. Clerk, financial expert, and credit expert are sample roles in this context. While the data entering is typically done by clerks, the decision on granting credits is done by financial experts, capable of assessing the risks involved in granting the credit.

2.2.5 Operational Aspect

The integration of existing tools and application programs into workflow applications is an important feature of workflow management systems. The information required is specified in the operational aspect of workflow schemas. The operational aspect covers mainly technical issues, like the invocation environment of external application programs (including host and directory information of the executable program as well as access rights), the definition of the input and output parameters of the application program and their mapping to input and output parameters of the respective workflow activities.

In this context the term integration refers to a rather loose coupling of the external application programs and the workflow management system. Typically the workflow management system provides the input parameters and invokes an external application. When it terminates, the result values of the program invocation are passed to the workflow management system, which can use this information during the remainder of the workflow. Therefore, external applications are also called invoked applications [118]. As will be discussed later in more detail, a closer integration of application-specific data and functionality on the one hand and workflow management functionality on the other hand

is feasible, using business objects in the application side and object technology in the workflow management side.

As indicated above, the integration of external applications in the workflow is not an easy task. This is mainly due to the fact that typically external applications were developed and used in an organization before workflow technology came in. Consequently, these applications are often not aware of the requirements imposed by workflow management systems. For instance to provide external applications with parameter values, invoke them and collect return values from then requires defined call interfaces, which — unfortunately — are not generally available. In this situation a specific software layer has to be developed to hide the details of the legacy system and to provide an interface for the workflow management system to that particular legacy system. This software layer is called wrapper. Wrappers can be implemented with middleware technology, for instance distributed object-oriented middleware.

In the banking example, a variety of information systems are used to perform specific tasks during the application process. Entering customer data and credit request data by clerks is typically done with forms-based software systems as front-ends of an integrated data repository run by the bank. The activities of assessing the risk of a credit may involve other information systems, some of which may reside in remote locations. An example of a remote information system is a centralized database which holds information on the credits in other banks the customers was granted already. In this case, the execution environment includes detailed information which allows the workflow management system to invoke the desired applications in the respective sites, using different kinds of middleware technology.

2.2.6 Flexibility Aspect

Flexibility has been an important aspect in workflow management from the beginning. One form of flexibility in this context is the ability to change the application process without re-coding application programs, simply by restructuring the activities and their execution constraints. Once a change to the workflow schema is made, all future workflow instances will use the new workflow schema. This flexibility aspect is satisfied by most workflow management systems.

As was already motivated in the Introduction, it was soon realized that this understanding of flexibility is too limited to cope with the requirements of advanced application processes in different application areas [27, 105, 84]. Starting from applications in non-traditional domains like the natural sciences, hospital environments, or engineering, flexibility also became a major issue in advanced business applications. This observation

is not surprising, since today's markets are evolving fast, and the rapid development of new application processes or the improvement of existing ones represents a major advantage for an enterprise.

Providing flexibility for workflow applications is based on the understanding that during workflow modeling not all aspects of the application process and its execution environment can be specified completely. There may be unforeseen situations during workflow executions, which require flexible reactions by the user or administrator of the system. If a workflow management does not support flexible reactions to unforeseen situations then its usability is restricted. As a result, measures are taken by workflow participants outside of the workflow management system to deal with the situation, which of course is not desirable. In contrast, the workflow management system should support the flexible reaction to unforeseen situations, so that the workflow instance can be continued under the supervision of the system. We believe that the future success of workflow management systems to a large extent depends on the way workflow schema and workflow instance modifications or changes to the organizational or technical environment of the workflow are supported in a user-friendly way.

There are different forms of flexibility, ranging from the change of role information and application program information to the change in the functional and behavioral aspects of workflows. Adding a workflow activity to a complex workflow while the workflow executes corresponds to a dynamic modification in the functional aspect; changing the control structure of sub-workflows of a given workflow (e.g., parallel execution of workflow activities, defined to be executed sequentially in the original workflow schema) corresponds to a modification in the behavioral aspect. A change of role information and of application program information changes the organizational and operational aspects, respectively.

Although the general structure of the sample credit request workflow is static, i.e., does not change frequently, numerous unforeseen events may occur during workflow executions, which require flexible reactions. For instance, assume while a credit request is processed, the applicant comes into an inheritance. This changes the financial situation of the applicant considerably, which may require a re-evaluation of the credit request. Simpler forms of flexibility occur when it comes to changes in role information or in application programs used to process workflow activities.

2.3 Workflow Languages

Generally, workflow languages aim at capturing workflow-relevant information of application processes and the environment in which application processes are executed, with the aim of their controlled execution by a workflow management system [108, 89, 32, 98]. Workflow languages are yet another species of languages for human-computer interaction. In contrast to general-purpose programming languages like Java or C++, workflow languages are highly domain specific, i.e., they are tailored towards the specific needs of workflow applications. Computational completeness is typically not an issue in workflow language design, since workflow languages are used to specify application processes, not computations in general. Since workflow schemas are used as an information basis for the modeling and optimization of application processes, graphical languages play an important role.

As discussed above, the components of workflow schemas and their relationships are modeled in workflow meta schemas. Hence, the constructs a workflow language provides are governed by the respective workflow meta schema. Obviously, given a workflow meta schema, there are potentially numerous languages which allow the specification of workflow schemas according to that particular workflow meta schema. These languages may or may not provide a graphical notation, and multiple graphical workflow languages may use different symbols for, e.g., workflows and execution constraint defined in the workflow meta schema. Workflow languages are designed to cover the workflow aspects discussed above. Depending on the target application domain of a workflow language, individual workflow aspects can be omitted and other aspects can be added. However, it is hard to imagine a useful workflow language which lacks expressive means for either of the functional, operational, organizational, and information aspects.

Modeling processes composed of concurrent or sequential activities has long been studied in the distributed computation discipline. In particular, a set of rigorous and mathematically founded approaches have been developed, among which process algebras play a key role, namely to formally define concurrently executing processes and their communication behavior. Important approaches include CCS [73] and CSP [49]. These approaches focus mainly on formal properties of distributed computations; distributed computations consist of a set of processes, which do not share memory and communicate by sending and receiving messages. In CSP, for instance, processes have specific communication commands, called guards, and there are communication channels between processes. Guards are attached to channels, to send a message to the channel or to take a message from it. Communication is generally assumed to be instantaneous, and guards can be selected in a non-deterministic fashion. Distributed algorithms can be formalized,

and their properties can be analyzed with CSP, for instance distributed synchronization algorithms. However, CSP is not well suited as a language to specify workflow schemas, since it considers a different level of abstraction, i.e., the implementation level of distributed computations. Since CSP and CCS were not invented to describe workflows, abstraction concepts suitable for workflow management are not present in these formal languages. Since furthermore technical and organizational aspects cannot be represented in these calculi adequately, they are not considered adequate as workflow languages. Therefore, they are not investigated further in this thesis.

Petri nets have widely been used to specify the behavior of a dynamic system with a fixed structure. They have also been used to model workflows. In a section on related work, Funsoft nets are described, an extended Petri net formalism tailored towards the needs of workflow management. An important class of workflow languages are graph-based languages, which allow the specification of workflows using different forms of directed graphs. While the functional and behavioral aspects can be specified adequately using graph notation, the information and operational aspects require additional mechanisms, like data types of application objects or information on the execution environment of external application programs. This information can be provided textually, often supported by workflow management systems using graphical user interfaces. Besides process graphs and Petri nets, script languages or workflow programming languages are used. These languages are typically closely related to particular workflow management system implementations. These systems often provide a graphical workflow language on top of the workflow programming language and the respective graphical user interface for workflow modeling. Workflow programming languages are also used as textual representations, for instance to export or import workflow schema information.

2.4 Summary

In this chapter basic concepts in workflow management are introduced. Based on workflow terminology, a sample application process is introduced informally. Workflow aspects are discussed as an important way to structure, to reduce the complexity, and to enhance the maintainability of workflow specifications. Ideally, each aspect in a workflow schema can be changed without having to adapt the other workflow aspects. This orthogonality of workflow aspects, however, cannot be offered in general. However, workflow aspects provide an important mechanism to cope with complexity in workflow specifications, and they allow flexible changes to individual parts of workflows.

Chapter 3

Specification of Workflows

This chapter introduces specifications of workflows in a step-wise way, guided by workflow aspects. After an informal discussion of workflow specifications based on directed graphs, a formal characterization of a workflow meta schema is presented. A sample workflow schema and a set of sample workflow instances based on that workflow schema will illustrate the concepts and formalisms introduced.

3.1 A Graph-Based Workflow Language

As shown in the previous chapter, workflow aspects are well suited to separate different dimensions in workflow specifications. As will be shown in this section they are also adequate for structuring an informal presentation of a graph-based workflow language, whose components are defined by a workflow meta schema. The following paragraphs discuss how the traditional workflow aspects are covered by the graph-based workflow language; the flexibility aspect will be covered below in detail.

3.1.1 Functional Aspect: Workflow Schema Trees

The functional aspect of a workflow schema describes what has to be done during the execution of a workflow and how the work is divided into sub-units of work, known as functional decomposition of workflows. In the workflow meta schema presented in this section, workflow schemas are based on directed graphs, whose nodes represent workflow schemas and whose edges represent constraints between these workflow schemas. Workflow schemas can be atomic or complex; while atomic workflows do not have an internal structure from a workflow management point of view, complex workflows consist of a set of workflow schemas, each of which can be atomic or complex. Hence, the

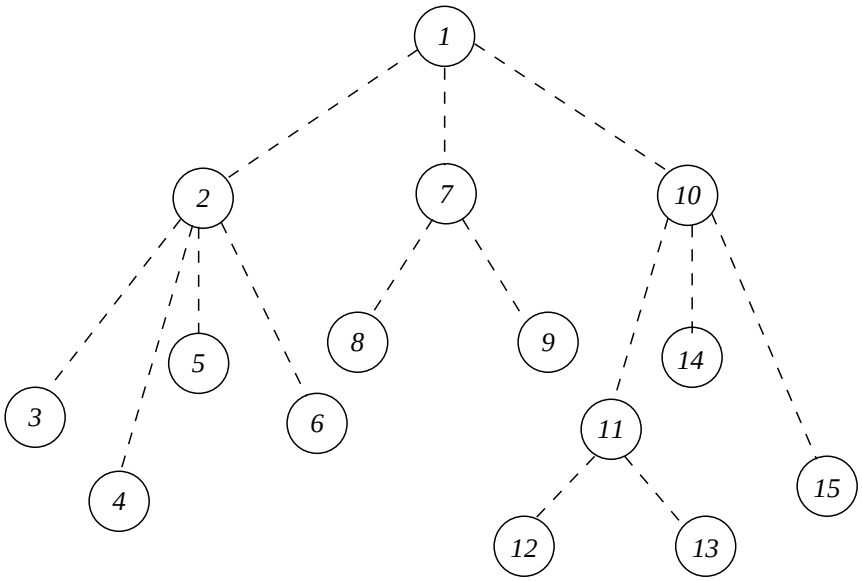

Figure 3.1: Workflow Schema Tree.

functional decomposition of workflow schemas is represented by a hierarchical structure called a workflow schema tree. A workflow schema tree can be of arbitrary depth; it is structured as follows: The root represents the top-level workflow, the inner nodes represent other complex workflows, and the leaves of the workflow schema tree represent atomic workflows.

A sample workflow schema tree is shown in Figure 3.1; the example is abstract, i.e., it does not reflect real world tasks like checking a credit request or ordering a set of books. However, the abstract example is well suited to introduce the concepts used in the workflow meta schema. The sample workflow schema tree consists of a number of nodes, representing complex and atomic workflow schemas. The root of the workflow schema tree is called top-level workflow schema. In the example, the top-level workflow schema 1 is decomposed into workflow schemas 2, 7, and 10. Due to their relative position, these are called sub-workflow schemas of workflow schema 1. Analogously, 1 is super-workflow schema of workflow schemas 2, 7, and 10.

The tree structure of workflow schemas implies that each workflow schema node except for the root has exactly one super-workflow schema. The inner nodes of a workflow schema tree represent complex workflow schemas, and each complex workflow schema has one or more sub-workflow schemas. In the example, 1, 2, 7, 10, and 11 are complex workflow schemas. Leaf nodes of the workflow schema tree represent atomic workflow schemas. Sub-workflow schemas of a given complex workflow schema are called sibling workflow schemas. For example 3, 4, 5, and 6 are sibling workflow schemas, as are 11,

14, and 15. (Notice that complex and atomic workflow schemas can be siblings.) The height of a workflow schema tree is the maximal length of a path connecting the root to a leaf node.
The hierarchical structure of a complex workflow schema defines a set of possible relationships between the workflow schemas involved. In general, each workflow schema can have relationships with their respective sibling workflow schemas and with its super-workflow schema only. No relationships are possible with workflow schemas of other branches of the workflow schema tree. This encapsulation of workflow functionality in accordance with the structure of workflow trees is an important property of workflow modeling, since it supports the principle of locality: Local changes to workflow schemas can be performed independently of changes in other parts of the workflow schema. For example, modifications local to sub-workflow schema 2 do not affect complex workflow schemas 7 or 10, which are the roots of the other branches of the complex top-level workflow schema. Just like in object-oriented programming (or in abstract data types), the structure and behavior of an object is specified by its signature, and the implementation of an object may change as long as its signature does not change. Workflow schemas will be formally specified later in this section.

3.1.2 Information Aspect: Parameters and Data Flow

The information aspect is based on parameters. Each workflow schema is assigned a set of typed input parameters and a set of typed output parameters. In general, when starting a workflow, the input parameters of the workflow are read. On the termination of a workflow, its results are written into the output parameters of the workflow.

From a technological perspective, workflow parameters are similar to parameters of procedures in traditional programming. As far as atomic workflows are concerned, this analogy is fairly accurate, since atomic workflows are implemented by external application programs. To execute an atomic workflow, its input parameters are transferred to input parameters of the external application. After the termination of the external application, its return values are transferred to the output parameters of the atomic workflow. To allow a change of the operational aspect, i.e., to substitute an external application A with an external application B without a change in the usage of the atomic workflow, the parameters of the workflow are mapped to the parameters of the application program used to implement the workflow. As long as the parameter on the workflow level do not change, the application program used to implement a particular atomic workflow can be changed. By providing a mapping from the input parameters of the workflow to the input parameters of the external application and from the output parameters of the application

to the output parameters of the workflow, this capability can be provided. If, however, application B requires new input parameters which are not provided by the atomic workflow then the workflow has to be changed in order to provide the missing values. In this case the interface of the workflow has changed, which means that additional modeling operations have to be performed in order to satisfy the parameter requirements of the new application program.

Complex workflows are performed by a set of sub-workflows, each of which can be atomic or complex. However, a complex workflow can also have parameters, whose values are received from or transferred to parameters of sibling workflows of the complex workflow, or to its sub-workflows. To this respect, complex workflow are similar to atomic workflows. While the latter deliver parameter values to application programs and receive parameter values from application programs, complex workflows deliver parameter values to sub-workflows and get parameter values from sub-workflows. Just like an atomic workflow schema, a complex workflow schema can be changed in its internal structure. While an atomic workflow schema can only be changed by changing the application program used to implement it, a complex workflow schema can be changed, for instance, by adding sub-workflow schemata to or by purging sub-workflow schemata from it. Due to the encapsulation of workflows according to the hierarchical structure of the workflow schema tree, local changes are not visible to other parts of the top-level workflow and can therefore be conducted without changing other parts of the workflow. If, however, the parameters of a complex workflow change then a new workflow schema has to be created, and the embedding of that new workflow schema in the context of one or more complex workflow schemas has to be reconsidered.

As indicated above, the transfer of information through the workflow is modeled by data flow, whose definition is based on parameters. Each workflow schema has a set of typed input parameters and a set of typed output parameters. There are two forms of data flow, one of which occurs between sibling workflows and the other of which occurs between a complex workflow and its sub-workflows. The former is called horizontal data flow; it is used to explicitly model the flow of information through the application process. The latter form of data flow is called vertical data flow; it can be used by a complex workflow to provide information to its sub-workflows, or to receive information generated during the execution of its sub-workflows.

In graphical representations of workflow schemas, input parameters are drawn as small boxes on the left hand side of a workflow schema node. Each parameter is marked with its identifier. To keep graphical representations of workflow schemas simple, data types of parameters are typically not displayed. Figure 3.2 shows the parameters and data flow connectors of a complex workflow schema 7, which also occurs as a sub-workflow

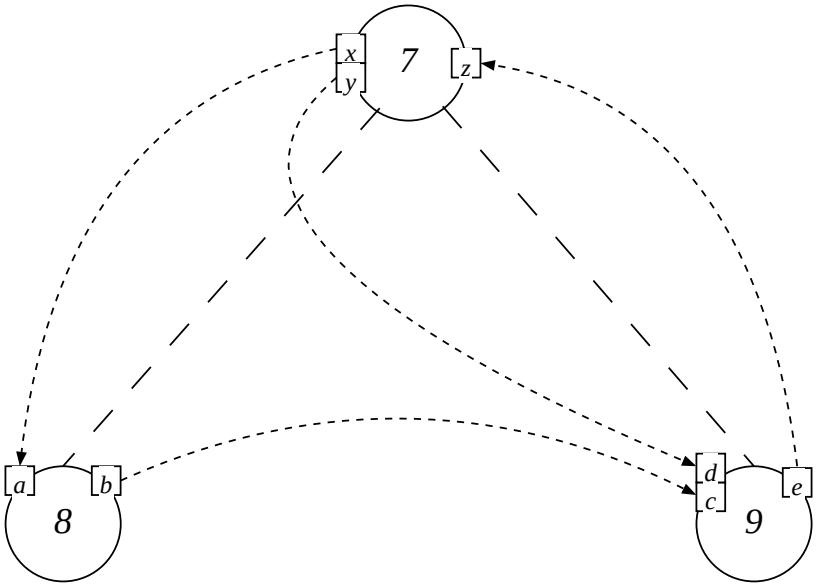

Figure 3.2: Horizontal and Vertical Data Flow.

schema in top-level workflow schema 1, shown in Figure 3.1. In Figure 3.2, for example, complex workflow schema 7 has input parameters x and y and an output parameter z, and atomic workflow schema 8 has an input parameter a and an output parameter b.
In Figure 3.2, data flow is represented by dotted arcs; the hierarchical structure of the workflow schema tree is represented by dotted lines. In the sample workflow schema, there is a horizontal data flow from output parameter b of workflow schema 8 to input parameter c of workflow schema 9. The complex workflow schema 7 has three vertical data connectors, two of which are oriented from the complex workflow schema to its sub-workflow schemas, while one vertical data flow provides the complex workflow with information generated during the execution of the sub-workflows. Due to their orientation in the workflow schema tree, these forms of vertical data flow can be characterized by "down" and "up". As shown in Figure 3.2, there are down-oriented vertical data connectors from input parameters x and y of complex workflow schema 7 to input parameters a of sub-workflow schema 8 and input parameter d of sub-workflow schema 9, respectively.

Figure 3.3 shows the complete workflow schema tree enhanced with data connectors, but without workflow schema parameters. Notice that there are horizontal data flow and vertical data flow constraints. As explained above, horizontal data flow occurs only between sibling workflow schemas, while vertical data flow occurs only between a complex workflow schema and its sub-workflow schemas. For example, there is horizontal data flow between sibling workflow schemas 3, 5, and 6, and there is vertical data flow between complex workflow schema 2 and its sub-workflow schemas 3 and 6, in orientations

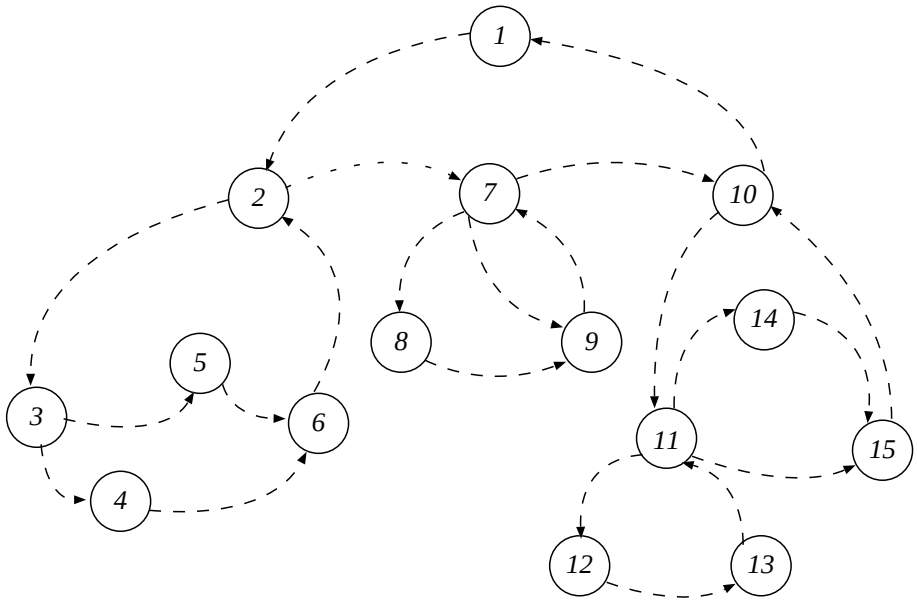

Figure 3.3: Workflow Schema Tree with Data Connectors (Hierarchy Edges Omitted).

"down" and "up", respectively.

In terms of workflow executions, we generally assume that input data parameters are read at the start of a workflow and that output parameters are written at the termination of the workflow. This means that no data flow can be defined between concurrent workflows. This definition of data flow between workflows implicitly defines an execution order of workflows. However, to separate workflow aspects, execution order dependencies and data dependencies between workflows are modeled separately. Execution order is covered in the behavioral aspect, discussed next.

3.1.3 Behavioral Aspect: Control Flow and Start Conditions

The behavioral aspect in workflow schemas specifies when and under which conditions workflows instances are being executed. In the workflow meta schema introduced, there are two constructs to define workflow behavior: Control flow constraints and start conditions. Sibling workflows can be related to each other by execution order constraints, expressed by control connectors. This form of execution constraint is also known as control flow.

Figure 3.4 shows the sample workflow schema enhanced with control connectors between sibling workflows; for clarity reasons, data flow is omitted in that figure. The execution constraints defined by control connectors specify that to execute top-level work-

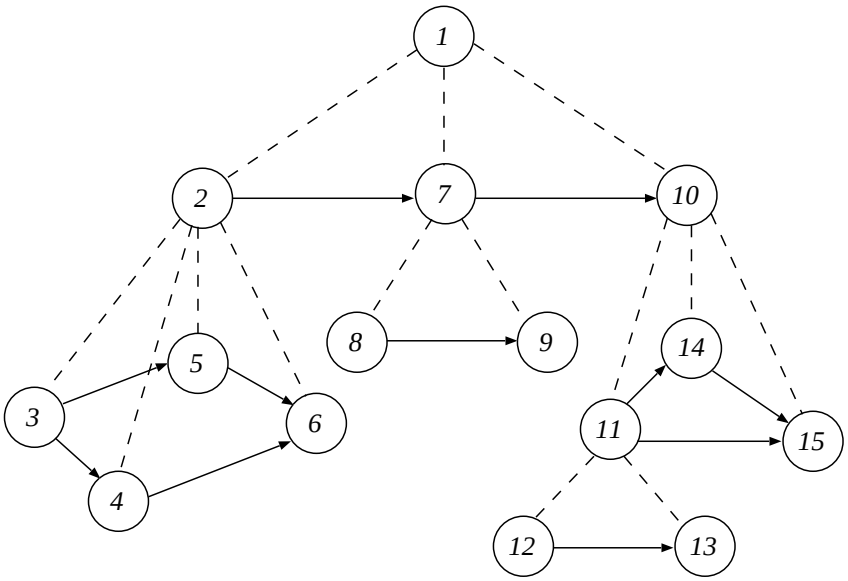

Figure 3.4: Workflow Schema Tree with Control Connectors.

flow 1, sub-workflow 2 has to execute first, which starts execution by initiating its sub-workflows recursively. In particular, workflow 2 is performed by executing workflow 3, followed by the concurrent execution of workflows 4 and 5. After these have terminated, workflow 6 can be performed, completing complex workflow 2. At this point in time, workflow 7 can start. Finally after 7 has terminated, workflow 10 can start. Sub-workflow schemas of a given complex workflow schema which are linked by control connectors can execute neither concurrently nor in arbitrary order: Their execution order is predefined by control flow constraints.

The conditions under which a given workflow instance is executed is defined by its start condition. By evaluating the start condition of a workflow, the system decides whether it has to be executed during a particular workflow instance. Clearly, a true value of the start condition makes the workflow execute, while the workflow is skipped if the start condition evaluates to false. To specify start conditions, workflow execution data as well as application-specific data can be used. In a credit request workflow, for instance, a grant credit workflow may have two alternative sub-workflows for different credit amounts requested. If the amount is less then a defined threshold then workflow one will be performed, otherwise workflow two will be executed. In this example the start condition of workflow schema one will check if the amount requested is less than the threshold, and the start condition of workflow two will check if the amount requested exceeds the threshold. This information can also be captured by transition conditions attached to control flow edges rather than by start conditions. However, start conditions are

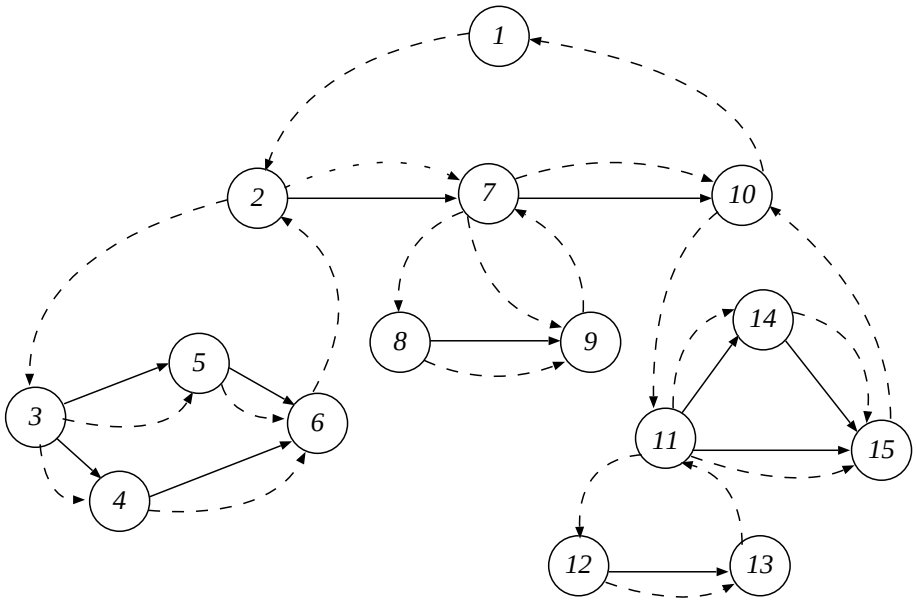

Figure 3.5: Workflow Schema Tree with Data Connectors and Control Connectors (Hierarchy Edges Omitted).

more general: Parameters which are transferred to workflows by data flow can be used in start conditions, like the credit amount in the sample workflow. In addition, start conditions allow to express alternative branches in workflow schemas by assigning condition C to one sub-workflow and condition $\neg C$ to the alternative sub-workflow. We used this approach in the previous example to model alternative execution of workflows one and two, respectively.

While control connectors are used to define potential execution order between sibling workflows, start conditions are used to determine whether a workflow is being executed. The term potential execution order indicates that a control connector from workflow i to workflow j does not necessarily mean that i and j will be executed — this is up to the start conditions. It means that if i and j are executed then the execution of j can only start after the termination of i. To summarize these considerations, Figure 3.5 contains the workflow schema tree with control flow connectors and data flow connectors.

3.1.4 Organizational and Operational Aspects

Workflow schemas introduced so far do not represent the workflow aspects discussed above entirely: While the functional, behavioral and information aspects are represented quite well, the organizational and operational aspects are not.

The organizational aspect of a workflow schema is used to specify the organizational environment in which the workflow instances will be executed. As discussed above, workflow administrators, workflow modelers and workflow participants are groups of persons which are involved in workflow executions. Typically the group of workflow participants are refined by additional roles, which model competences, skills and availabilities of persons responsible for the execution of workflow activities. Organizational information can be added to workflow schemas in textual form. However, if the organizational structure of an enterprise is maintained by an information system then appropriate interfaces have to be provided between the information system and the workflow management system in order to avoid redundant management of organizational information and to use the information on the organizational structure for the purposes of workflow management.

To investigate the organizational aspect, a set of roles, a set of persons, and a set of external application programs is introduced. Roles characterize persons with similar capabilities and competences. “Clerk” and “project manager” are examples of roles. In addition, there are roles for describing administration of and responsibilities for workflows. Examples are workflow administrator and workflow supervisor. While workflow administrators are responsible for the operation and maintenance of workflow management systems, workflow supervisors are responsible for the correct and timely execution of particular workflows instances, assigned to them. Typically, a workflow supervisor is a manger of the department in which the workflow is processed. In real world applications, other roles can be added according to the requirements of the workflow application.

3.2 Formalization

Based on the informal introduction to graph-based specification of workflows, this section presents a formal specification of workflow schemas and workflow instances. While a formal specification is a useful basis for describing the domain of interest in detail, it clearly is not sufficient to design and implement workflow management systems. Therefore, Part II introduces an object-oriented design of a workflow management system, and this design makes extensive use of the formalization of workflows, discussed in the remainder of this section.

This section is organized as follows: First, workflow schemas are formalized, starting with workflow schema nodes and proceeding towards workflow schema graphs. A workflow schema graph is a representation of a complex workflow schema; each workflow schema graph consists of a set of workflow schema nodes, each of which may represent

an atomic workflow schema or a complex workflow schema. The same approach is taken to formalize workflow instances: Based on a specification of workflow instance nodes, workflow instance graphs will be described formally. The execution of workflow instances will be specified in an algorithmic form. The notations and algorithms introduced will be illustrated in a sample workflow, which is based on the credit request application process sketched above.

3.2.1 Workflow Schemas

The basis of the formalization of workflow schemas is provided by workflow schema nodes. Depending on their particular context, workflow schema nodes may characterize atomic workflow schemas or complex workflow schemas. Since on a given level of abstraction there is no distinction between atomic and complex workflow schemas, both kinds of workflow schemas can be formalized by the notion of workflow schema nodes.

As an example, reconsider Figure 3.4, which shows the hierarchical structure of a top-level workflow schema 1. Its immediate sub-workflow schemas are complex workflow schemas 2, 7, and 10, each of which can be represented by a workflow schema node. In particular, workflow schema 10 can be represented by a workflow schema node, defining, for example, its input and output parameters and its start condition. When it comes to representing the internal structure of workflow schema 10, it has to be refined according to the structure shown in Figure 3.4. As a result, its workflow schema graph consists of workflow schema nodes 11, 14, and 15 with their respective control flow and data flow constraints. This abstraction concepts is valid in the context of the graph-based workflow language introduced, since complex workflow schemas are encapsulated, i.e., there are no control flow or data flow constraints between a sub-workflow of a given complex workflow and a workflow of a different branch of the workflow schema tree.

After the discussion of the general approach to formalizing workflow schemas, some notation is introduced. As indicated above, parameters of workflows are used to capture the information aspect in workflow schemas. As a basis for a formalization of this aspect, the set of data types is represented by T. $Dom(t)$ refers to the domain of data type $t \in T$, i.e., it represents the set of values which variables of type t can take. The set T contains standard data types (e.g., integer, float, string) as well as specific data types, required for particular applications (e.g., Order, Invoice, Person). Hence, T is extensible: Depending on the application, data types can be added to or deleted from T.

To represent data flow between workflows, workflow schemas are assigned parameters. The set of parameters of a workflow schema is partitioned in a set of input parameters, a set of start input parameters, and a set of output parameters. Besides data flow,

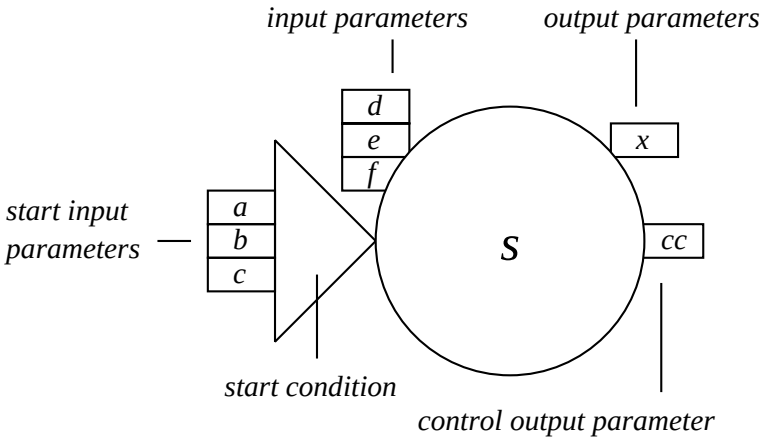

Figure 3.6: Individual Workflow Schema Node, with Parameters and Start Condition.

control flow is also represented by parameters; these parameters are called control (input or output) parameters, and they are of type CCType (control connector type), such that

$$Dom(\text{CCType}) \;:=\; \{\text{true-signaled, false-signaled}, \perp\}$$

To separate workflow-relevant data types from application-specific data types, we stipulate that CCType $\notin T$. The domain of CCType includes $\perp$, which represents the undefined value. This value is used to indicate that a control connector is not yet signaled. In general, the control connector of a workflow instance is signaled only after it terminates or, as will be discussed shortly, after the decision is taken that the workflow instance will not be executed in a particular case. In general, parameter values can even be left blank permanently, e.g., if the respective workflow will not be executed. To capture this situation, the undefined value is included in the domains of application data types as well, i.e., $\perp \in Dom(t), \forall\, t \in T$. TypeOf$(p) \in T \cup \{\text{CCType}\}$ refers to the data type of a parameter p.

Definition 3.1 A *workflow schema node* s consists of

- a set in_s of typed input parameters, such that TypeOf$(p) \in T,\ \forall\, p \in in_s$,
- a set out_s of typed output parameters with one and only one dedicated control output parameter $cc \in out_s$, such that TypeOf$(cc) =$ CCType. The other output parameters have application-specific data types: TypeOf$(q) \in T$ for all $q \in out_s$, $q \neq cc$,
- a list $sip_s = (p_1, p_2, \ldots, p_{n_s})$ of typed start input parameters, partitioned in a set of control input parameters p, such that TypeOf$(p) =$ CCType, and a set of control flow relevant data parameters q, such that TypeOf$(q) \in T$,

- a start condition sc_s, such that

$$sc_s : dom(t_1) \times dom(t_2) \times \ldots \times dom(t_{n_s}) \mapsto \{true, false\}$$

if $sip_s = (p_1, p_2, \ldots, p_{n_s})$ is a list of start input parameters and TypeOf$(p_i) = t_i$ for $1 \leq i \leq n_s$.

$\diamond$

Notice that control parameters are not permitted as input parameters, characterized by TypeOf$(p) \in T,\ \forall\, p \in in_s$, and CCType $\notin T$ in Definition 3.1. Control parameters may only occur as parameters of start conditions, i.e., as start input parameters.

While the formal characterization of workflow schemas represents control flow using the data flow construct of parameters, workflow modeling separates these two aspects: Workflow modelers use control flow to indicate execution order constraints and data flow to indicate the flow of information during workflow executions. In workflow modeling, the separation of workflow aspects is important to allow workflow modelers to define independently and in a modular way, i.e., execution order constraints and information constraints. When the design and development of workflow management systems is introduced in Part II, this aspect is discussed in more detail. In particular, a graphical workflow modeling tool is presented which supports the separate definition of control flow and data flow. On a conceptual level, however, representing control flow by parameters is an adequate choice, since a formalism for parameters is required to represent data flow anyhow, and no additional concepts have to be introduced to capture control flow.

Given a parameter a, $\nu(a) \in Dom(t)$ refers to its value, if TypeOf$(a) = t \in T \cup \{\text{CCType}\}$. Referring to the value of a parameter a by, for instance, $\nu(a) = 100$ is more precise than by $a = 100$, since it allows to specify that a is an input parameter of workflow schema s by $a \in in_s$, and it allows to specify that the values that a can take are in the domain of t by $\nu(a) \in dom(t)$. Referring to the value of parameter a simply by $a = 100$ would contradict $a \in in_s$. The requirement for this notation results from the fact that in a single formalism two levels of abstraction have to be covered, i.e., the type level of workflow schemas and the instance level of workflow instances.

Figure 3.6 shows a workflow schema node i with parameter sets and a start condition. The start condition of a workflow schema node is graphically represented by a triangle on its left hand side with start input parameters attached to it. Input parameters are shown on the upper left hand side, and output parameters on the upper right hand side of workflow schema nodes; the control output parameter of a workflow schema node is drawn on its right hand side. As can be seen in that figure, a, b, c are start condition parameters, while d, e, f and x, cc are input parameters and output parameters, respectively.

The definition of workflow schema nodes in general and the workflow schema node s shown in Figure 3.6 in particular is now illustrated with the credit check workflow schema as part of the credit request application process, as discussed above: The start input parameter a is of type integer, while b and c are of type CCType. Given this typing of parameters, a typical start condition is

$$sc_s \equiv \nu(a) < 20 \wedge \nu(b) = \text{true-signaled} \wedge \nu(c) = \text{true-signaled}$$

This start condition indicates that the execution of the workflow depends on the termination of two previous workflows (represented by $\nu(b) = \text{true-signaled} \wedge \nu(c) = \text{true-signaled}$) and on the fact that an application-specific value is less than a given threshold. Input parameter d is of type integer, while e and f are of type string, illustrating the use of application-specific information required for the execution of the workflow. The output parameter x indicates the generation of an application-specific output value, for instance a decision string ("y", "n"), representing the granting or rejection of the credit request. The value of the control output parameter cc indicates whether the workflow was executed during a particular workflow instance.

Based on the definition of workflow schema nodes, the following definition characterizes workflow schema graphs, consisting of a set of workflow schema nodes and constraints between them. At this point we assume that the sub-workflow schemas of a given complex workflow schema are different, i.e., each workflow schema occurs at most once as a sub-workflow schema in a given complex workflow schema. This restriction will be lifted in Part II when an object-oriented design of a workflow management system is introduced.

Definition 3.2 A *workflow schema graph* G_s as a refinement of a workflow schema node s is defined by $G_s = (V_s, C_s, D_s)$, such that

- V_s is a set of workflow schema nodes, $s \in V_s$
- C_s is a set of control connectors between parameters of sibling workflows:

  $$C_s \subseteq out_j \times sip_k$$

  such that $j, k \in V_s - \{s\}$, and the parameters involved are of type CCType:

  $$(p, q) \in C_s \Rightarrow \text{TypeOf}(p) = \text{TypeOf}(q) = \text{CCType}$$

  The fact that $(p, q) \in C_s$ and $p \in out_j$, $q \in sip_k$ is graphically represented by a directed edge with source node j and destination node k. This edge is called control connector. In detailed graphical representations, the control connector edge connects parameters p and q directly.

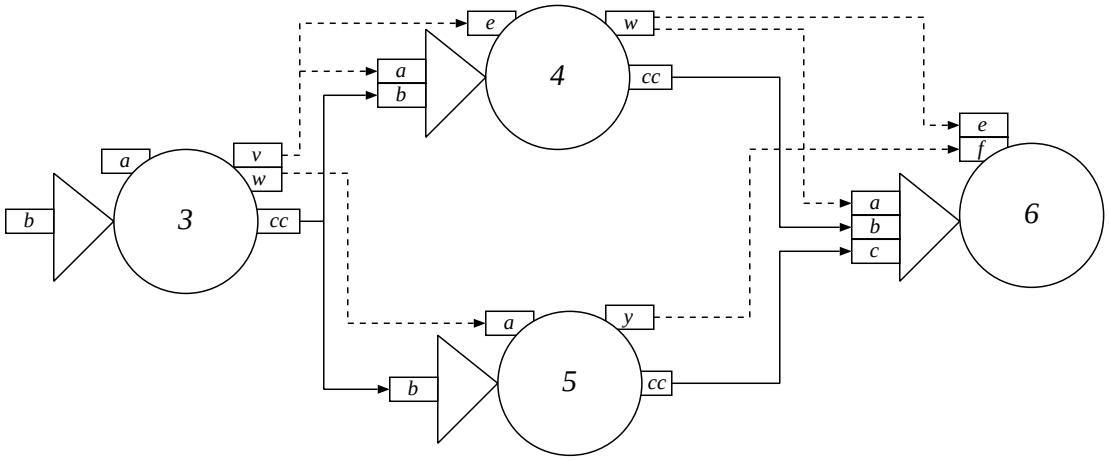

Figure 3.7: Detailed View of Complex Workflow Schema 2.

- D_s is a set of data connectors, such that D_s is partitioned in a set D_s^h of horizontal data connectors and a set D_s^v of vertical data connectors.

  Horizontal data connectors exist only between parameters of sibling workflow schema nodes, such that

  $$D_s^h \subseteq out_j \times (in_k \cup sip_k)$$

  for $j, k \in V_s - \{s\}$.

  Vertical data connectors can only occur between parameters of s and parameters of its sub-workflow schemas:

  $$D_s^v \subseteq (in_s \times (in_j \cup sip_j)) \cup (out_j \times out_s)$$

  for $j \in V_s - \{s\}$. The fact that $(p, q) \in D_s$, such that p is a parameter of j, and q is a parameter of k is graphically represented by a directed dotted edge with source node j and destination node k. This edge is called data connector. In detailed graphical representations, the data connector edge connects the parameters involved.

$\diamond$

As a pathological case, these definitions also allow to characterize the workflow schema graph for an atomic workflow schema s. In this case V_s only contains a single element, s. Hence, the workflow schema graph of an atomic workflow schema is defined by $G_s = (\{s\}, \{\}, \{\})$, which can be regarded equivalent to s.

Figure 3.7 shows the workflow schema graph of complex workflow schema 2, involving sub-workflow schemas 3, 4, 5, and 6, as well as parameters and control and data connectors. This workflow schema graph is represented by $G_2 = (V_2, C_2, D_2)$, such that

$$\begin{aligned} V_2 &= \{2, 3, 4, 5, 6\} \\ C_2 &= \{(3.cc, 4.b), (3.cc, 5.b), (4.cc, 6.b), (5.cc, 6.c)\} \\ D_2^h &= \{(3.v, 4.e), (3.v, 4.a), (3.w, 5.a), (4.w, 6.e), (4.w, 6.a), (5.y, 6.f)\} \\ D_2^v &= \{\} \end{aligned}$$

In order not to overload the figure, the workflow schema node for the complex workflow 2 and vertical data flow constraints are omitted. In workflow schema graphs, parameter identifiers are local within their respective workflow schema nodes. For unique identification within the context of a workflow schema graph, the parameter identifier is prefixed with the identifier of the respective workflow schema node. In the example, for instance, $(3.cc, 4.b)$ represents a control flow between sub-workflow schemas 3 and 4 involving the control output parameter cc of 3 and the control input parameter b of workflow schema node 4, while $(3.w, 5.a)$ describes a data flow from sub-workflow schema node 3 to sub-workflow schema node 5, involving output parameter w of workflow schema node 3 and input parameter a of workflow schema node 5.

3.2.2 Consistency Criteria for Workflow Schemas

Workflow modeling aims at capturing properties of the application domain and of particular application processes using workflow languages. Semantic correctness properties of application processes are highly domain-specific; these properties are defined by the specific policies or goals of the organization. There even may be conflicting goals, which need to be resolved during workflow modeling. For instance, the goal of fast response to client requests may conflict with the goal of reducing personnel cost. As a result, generic correctness criteria for workflow schemas cannot be provided by workflow technology. However, consistency criteria on the syntactic structure of workflow schemas can in fact be identified. It is assumed that workflow schemas generally satisfy these criteria; a sample criteria is motivated as follows.

The example is based on a complex workflow schema k with sub-workflow schemas i and j, a control flow $i \rightarrow j$, and a data flow between these sibling workflow schemas in the opposite direction. This situation is shown in Figure 3.8. As as result, i can never use the data specified by the data flow, since the data would only be written to the output parameter after j completes. On the other hand, there is an execution order constraint

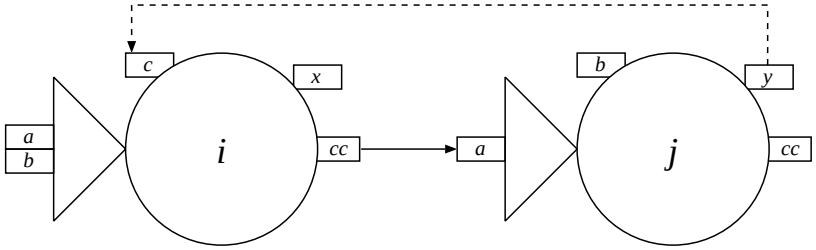

Figure 3.8: Conflicting Control Flow and Data Flow.

imposed on i and j by the control flow: j cannot be started until i terminates. Using a workflow schema with these properties to control workflow instances leads to a deadlock-like situation, in which two workflow activities mutually wait for each other to terminate: i waits for j to terminate since it requires the data generated by j, while j waits for i to terminate since due to the control flow constraint, j cannot begin its execution while i is not completed. To avoid situations like the one described, a set of consistency criteria are defined for workflow schemas.

Definition 3.3 Let s be a complex workflow schema with a workflow schema graph G_s; s is *consistent*, if the following criteria are satisfied:

1. *Completeness:* For each input parameter $q \in in_j$ of a workflow schema node $j \in V_s$ there is a data flow connector $(p, q) \in D_s$, such that $p \in out_i \cup in_s$, $i \in V_s - \{s\}$.

2. *Type compatibility:* Data flow connectors link parameters of the same type, i.e., for each data connector $(p, q) \in D_s$, TypeOf(p) = TypeOf(q).

3. *Data availability:* For each horizontal data connector $(p, q) \in D_s^h$, such that $p \in out_i$ and $q \in in_j \cup sip_j$, there is a path of control connectors from i to j. We say that data flow follows control flow.

4. *Acyclic structure:* Control connectors of a given workflow schema graph do not form cycles. Since data flow follows control flow, the structure of data flow constraints is also acyclic.

$\diamond$

We mention that recursion in workflow schemas is supported. This means that a workflow schema i can appear as a sub-workflow schema of itself, i.e., of workflow schema i. Since recursion and loops are concepts of equivalent expressiveness and unrestricted loops in workflow schemas amount to unstructured programming similar to the goto statement in

early programming languages, we opted for recursion instead. The use of recursion will be exemplified in the next section when executions of workflow instances are discussed.

The workflow schema shown in Figure 3.8 is not consistent, since the data availability constraint is not satisfied. In particular, there is a data connector $(y, c) \in D_s^h$, such that $y \in out_j$ and $c \in in_i$, but there is a no control connector from j to i.

In the context of dynamic adaptations of workflow instances, relationships between workflow schemas have to be taken into account. In order to provide a formal basis for these considerations (which will be discussed in Part II), the following definition considers prefixes of workflow schemas.

Definition 3.4 Let s and s' be complex workflow schemas with workflow schema graphs G_s and $G_{s'}$, respectively. Workflow schema s is a *prefix* of s' if and only if G_s is a subgraph of $G_{s'}$ (represented by $V_s \subseteq V_{s'}$ $C_s \subseteq C_{s'}$ $D_s \subseteq D_{s'}$) and the following holds: For each connector $(p, q) \in (C_{s'} - C_s) \cup (D_{s'} - D_s)$, such that p is a parameter of j and q is a parameter of k, k is not in the prefix V_s, i.e.,

$$k \in V_{s'} - V_s$$

If s is a prefix of s' then the workflow schema nodes in $V_{s'} - V_s$ are in the *postfix* of s' with respect to s; the workflow schema nodes in V_s are in the *prefix* of s with respect to s'. ◇

Hence, a workflow schema s is a prefix of a workflow schema s' if the workflow schema graph of s is included in the workflow schema graph of s'. In addition, no control connectors and no data connectors are allowed to have a workflow schema node of V_s as a target. The rational behind this definition is explained as follows: If in the postfix of the complex workflow schema s', a data flow is defined to lead into a workflow schema node of s then clearly s cannot be a prefix of s'. Analogously, an execution constraint leading from the postfix of s' into s is not meaningful, since this would mean that a workflow activity in the postfix would be executed before a workflow activity in the prefix.

Theorem 3.1 The consistency property of workflow schemas is prefix-closed, i.e., a prefix of a consistent workflow schema is again a consistent workflow schema.

Proof Let s be a prefix of a consistent workflow schema s', and $G(s)$ and $G(s')$ the respective workflow schema graphs. To prove that s is a consistent workflow schema graph, Definition 3.3 has to be reconsidered:

1. *Completeness:* (Proof by contradiction.) Assume the completeness property is violated in s. Without loss of generality, assume there is an input parameter $q \in in_j$ of a workflow schema node $j \in V_s$ for which there is no data flow connector $(p, q) \in D_s$, such that $p \in out_i \cup in_s$, $i \in V_s - \{s\}$. Since s' is consistent, such a data flow edge is present in $D_{s'}$. Hence, there is a connector $(p, q) \in D_{s'} - D_s$ such that p is a parameter of i, and q is a parameter of j, and $j \notin V_{s'} - V_s$, contradicting Definition 3.4. Hence, s cannot be a prefix of s', contradicting the assumption and proving the claim.

2. *Type compatibility:* Since no changes to data types of parameters are performed, type compatibility of s is correct, since s is a prefix of s' and s is a consistent workflow schema.

3. *Data availability:* For each horizontal data connector $(p, q) \in D_s^h$, such that $p \in out_i$ and $q \in in_j \cup sip_j$, there is a path of control connectors from i to j. This property is satisfied, since the prefix property as specified in Definition 3.4 determines that control connectors which link workflow instance nodes in the prefix cannot be deleted.

4. *Acyclic structure:* This property is satisfied, since a prefix of an acyclic structure has obviously also an acyclic structure.

Hence, a prefix of a consistent workflow schema graph is a consistent workflow schema graph. □

3.2.3 Workflow Instances

Analogously to workflow schemas, workflow instances are specified by defining workflow instance nodes and workflow instance graphs in turn, consisting of a number of workflow instance nodes and constraints between them. Since the internal structure of workflow instances is encapsulated, workflow instance nodes are a common abstraction for both complex and atomic workflow instances. To specify the workflow schema which a workflow instance is based on, we use the SchemaOf function: If a workflow instance i is based on a workflow schema s then SchemaOf$(i) = s$.

Without going into the details of workflow instance enactment (which will be discussed in the next section), Figure 3.9 shows one particular state during the execution of a workflow instance node i. This workflow instance is based on workflow schema node s shown in Figure 3.6, i.e., SchemaOf$(i) = s$. While workflow schema nodes maintain

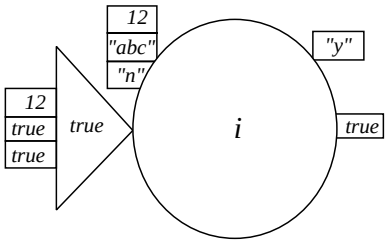

Figure 3.9: Workflow Instance Node i.

parameters as well as data type information, parameters of workflow instance nodes have values, as defined by the domains of the parameters.

The workflow instance node shown in Figure 3.9 represents a workflow instance, which is already completed, since not only its input parameters but also its output parameters contain values. In earlier stages of that workflow instance, this has not always been the case. In general, when the execution of a workflow instance starts, all its parameters are undefined. Only when the workflow execution proceeds or when data values are transferred to the workflow's parameters, the parameters are provided values. As indicated above, parameter values can even be left blank permanently, i.e., if a particular sub-workflow will not be executed, or temporarily, i.e., if the sub-workflow has not yet been executed. In both cases, the missing value is represented by the undefined value $\perp$. In graphical representations, the boxes of parameters with the undefined value are left blank. It is assumed that while the start condition $sc_s(sip_i)$ of a workflow instance i with a workflow schema s is not evaluated, it has the undefined value.

Definition 3.5 A *workflow instance node* i based on a workflow schema node s consists of

- a parameter p for each start input parameter in the list $sip_s = (p_1, p_2, \ldots, p_{n_s})$ of start input parameters of workflow schema node s and a value $\nu(p) \in Dom(t)$ for that parameter, if TypeOf$(p) = t$,
- a parameter p for each input parameter $p \in in_s$, TypeOf$(p) = t$, and a value $\nu(p) \in Dom(t)$,
- a parameter p for each output parameter $p \in out_s$, TypeOf$(p) = t$, and a value $\nu(p) \in Dom(t)$,
- a start condition $sc_s(sip_i) \in \{$true, false, $\perp\}$, based on the values of the start input parameters.

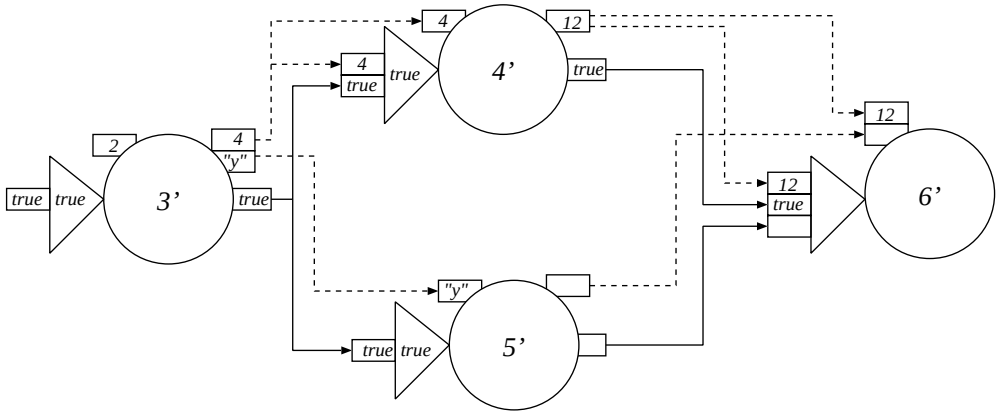

Figure 3.10: Workflow Instance Graph.

◇

If multiple workflow instances are discussed and from the context it is not clear to which workflow instance a particular parameter belongs then parameters are prefixed with the identifier of their respective workflow instance. For example, a parameter p of a workflow instance i is then characterized by $i.p$. Since workflow instance identifiers are unique, parameters are also unique.

Definition 3.5 is now applied to the sample workflow instance node shown in Figure 3.9. There are three input parameters d, e, f, such that $\nu(d) = 12$, $\nu(e) = "abc"$, and $\nu(f) = "n"$, respectively. The start input parameter values are $\nu(a) = 12$, $\nu(b) = "y"$, and $\nu(c) = true$ (as a shorthand notation for $\nu(c) =$ true-signaled). The start condition is evaluated to true, the workflow is executed, and the control output parameter is set to true, i.e., true-signaled. If the start condition was evaluated to false then the workflow instance would not have been executed. In this case the output parameters are left blank, except for the control flow output parameter which is assigned the value false-signaled to indicate that the workflow instance was not performed.

Proceeding to workflow instance graphs, Figure 3.10 is considered, which shows a snapshot of such a graph, consisting of workflow instance nodes $3', 4', 5', 6'$ and control flow and data flow connectors between them. In the situation shown, workflow instance nodes $3'$ and $4'$ have already terminated, since their output parameters contain values. $5'$ is currently executing, since the start condition is evaluated to true and the output parameters are still blank, representing the undefined value. Workflow instance node $6'$ has already received parameter values from $4'$. Since it still waits for $5'$ to complete, its start condition has not yet been evaluated, and consequently, it has not started yet.

Definition 3.6 A *workflow instance graph* $G_i = (V_i, C_i, D_i)$ based on a workflow schema graph $G(s) = (V_s, C_s, D_s)$ is defined as follows:

- For each workflow schema node $s' \in V_s$ there is a workflow instance node $i' \in V_i$, such that SchemaOf$(i') = s'$.
- For each control connector in C_s there is a control connector in C_i, linking the respective parameters. In particular, for each control connector $(i.cc, j.a)$ in C_s there is a control connector $(i'.cc, j'.a)$ in C_i, such that SchemaOf$(i') = i$ and SchemaOf$(j') = j$.
- For each data connector in D_s there is a data connector in D_i, linking the respective parameters. In particular, for each data connector $(i.a, j.b)$ in D_s there is a data connector $(i'.a, j'.b)$ in D_i, such that SchemaOf$(i') = i$ and SchemaOf$(j') = j$.

$\diamond$

Based on this definition, the sample workflow instance graph shown in Figure 3.10 is formalized by $G_{2'} = (V_{2'}, C_{2'}, D_{2'})$, such that

$$
\begin{aligned}
V_{2'} &= \{2', 3', 4', 5', 6'\} \\
C_{2'} &= \{(3'.cc, 4'.b), (3'.cc, 5'.b), (4'.cc, 6'.b), (5'.cc, 6'.c)\} \\
D^h_{2'} &= \{(3'.v, 4'.e), (3'.v, 4'.a), (3'.w, 5'.a), (4'.w, 6'.e), (4'.w, 6'.a), (5'.y, 6'.f)\} \\
D^v_{2'} &= \{\}
\end{aligned}
$$

Analogously to the workflow schema graph discussed above, the workflow instance node for the complex workflow instance (workflow instance node $2'$) and vertical data connectors are now shown in that figure. We mention that while in the formal characterization of workflow instance graphs the parameter names are given, in graphical representations, the values of the parameters can be shown instead. Hence, for each parameter $(i'.a)$ of a workflow instance graph, the value $\nu(i'.a)$ can be displayed.

3.2.4 Workflow Execution Semantics

To describe the execution semantics of workflow instances, the states and the state transitions that a workflow instance may take during its execution are specified by state transition diagrams. When turning to the object-oriented design of a workflow management system in Part II, the state transition diagram will be enhanced to cope with technical

issues and with different kinds of workflow instances, namely atomic and complex workflow instances. From a conceptual point of view, the behavior of atomic workflow instances and complex workflow instances can be described by a common state transition diagram.

The execution of a workflow instance is described as follows: In general, an application-specific event triggers the initiation of a workflow instance, based on a workflow schema. For example, a credit request put forward by a customer triggers the instantiation of a workflow instance based on a credit request workflow schema. That workflow schema is associated with the triggering event, i.e., the receipt of a credit request.

Assuming the structure of an application process is represented by a complex top-level workflow schema s, the instantiation and controlled execution of a workflow instance based on that workflow schema is described as follows: A top-level workflow instance i is instantiated, according to workflow schema s. The structure of that workflow instance node is specified in Definition 3.5. To keep track of the relationship between the workflow instance and the workflow schema, SchemaOf$(i) = s$. For each sub-workflow schema of s, a sub-workflow instance of i is instantiated, as are associated objects, representing control flow and data flow constraints between them as well as the start conditions of the sub-workflow instances. At this point in time, the workflow instance node i is refined with a workflow instance graph $G(i)$, as specified in Definition 3.6.

When all control input parameters of a given sub-workflow are set, i.e., when all workflow instances which precede the workflow instance in control flow have terminated, the sub-workflow instance enters the ready state. In this state, the start condition is not yet evaluated, but it is ready for evaluation. The next state transitions are triggered by the result of the start condition: If it evaluates to true then the workflow instance enters the running state; if the start condition evaluates to false, the workflow instance enters the eliminated state. In general, alternative branches of a complex workflow which are not performed are eliminated. In the credit request workflow, for instance, granting the credit request and rejecting it are alternative branches. If a particular credit request is granted then the reject branch of the workflow instance needs to be eliminated, since it will not be executed in that particular workflow instance. Once a running workflow instance is completed, it enters the done state. This discussion of the life cycle of a workflow instance is captured in the state transition diagram shown in Figure 3.11.

The state transition diagram provides the basis for the next steps in specifying the execution semantics of workflow instances, in which for each state a precondition will be defined. A workflow instance may enter a particular state only if it satisfies the precondition of that state. The definition of preconditions will be based on parameter values of workflow instances. In a second step, algorithms for executing workflow instance nodes

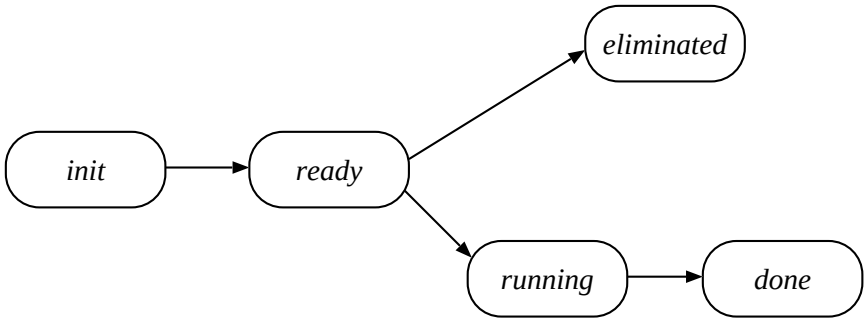

Figure 3.11: State Transition Diagram for Workflow Instances.

and for executing workflow instance graphs will be presented. The following definition specifies the conditions that hold when a workflow instance i with SchemaOf$(i) = s$ enters specific states, as defined by the state transition diagram. The state of a workflow instance is defined by the StateOf function, such that

$$\text{StateOf}(i) \in \{\text{init, ready, eliminated, running, done}\}$$

Definition 3.7 The preconditions of states of workflow instances are defined as follows:

In the init state, a workflow instance is created and gathers values for its input parameters and its start input parameters:

$$\text{StateOf}(i) = \text{init} \quad :\Leftrightarrow \quad (\exists p \in sip_i)\ \text{TypeOf}(p) = \text{CCType} \wedge \nu(p) = \bot$$

To enter the ready state, all control input parameters have to be set, i.e., all sibling workflow instances which precede the workflow according to control flow as specified in workflow schema s have to be completed or eliminated. In the ready state, the start condition is not yet evaluated, but is ready for being evaluated.

$$\text{StateOf}(i) = \text{ready} \quad :\Leftrightarrow \quad \nu(p) \neq \bot \wedge sc_s(sip_i) = \bot,\ \forall p \in sip_i, \text{TypeOf}(p) = \text{CCType}$$

The evaluation of the start condition triggers a state transition. A true value triggers the transition to the running state, while a false value puts the workflow instance in the eliminated state. The precondition of the running state is defined by

$$\text{StateOf}(i) = \text{running} \quad :\Leftrightarrow \quad sc_s(sip_i) \equiv true \wedge \nu(p) = \bot,\ p \in out_i$$

The eliminated state has to take into account the values of the output parameters of the workflow instance, which are set by now:

$$\begin{aligned}\text{StateOf}(i) = \text{eliminated} \quad :\Leftrightarrow \quad & sc_s(sip_i) \equiv false \ \wedge\ cc = \text{false-signaled} \ \wedge \\ & \nu(p) = \bot,\ \forall p \neq cc \in out_i\end{aligned}$$

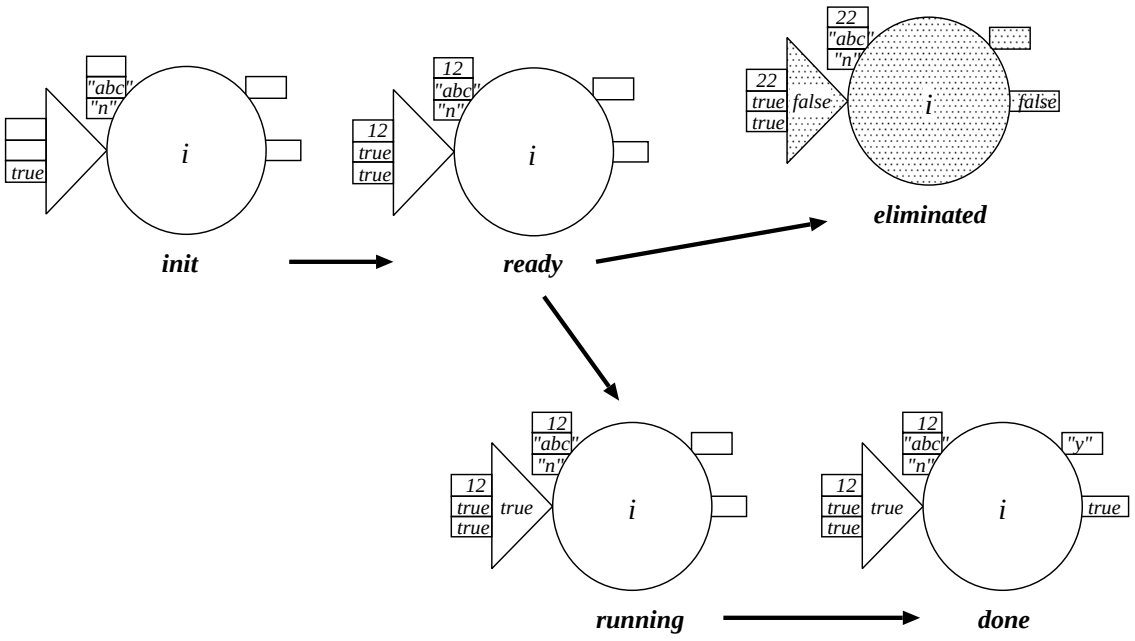

Figure 3.12: States of Workflow Instance Node i.

It is stipulated that after finite time, a running workflow instance terminates. The termination event triggers a transition to the done state. In this state, all output parameters are set, and the value of the control output parameter indicates that the workflow instance has been executed:

$$\text{StateOf}(i) = \text{done} \quad :\Leftrightarrow \quad \nu(p) \neq \perp \wedge \nu(cc) = \text{true-signaled}, \; \forall p \in out_i$$

$\diamond$

To illustrate these properties, Figure 3.12 shows a workflow instance node i in different states during its execution. In the left hand side of that figure, the workflow instance node is in the init state. One predecessor workflow instance (not shown) has already terminated, and a true-signaled value (represented by *true* in Figure 3.12) is transferred to a start input parameter. In addition, application-specific data values are transferred to the input parameters of that workflow instance. Since not all control input parameters are filled yet, the start condition cannot be evaluated at this point. The workflow instance enters the ready state if and when all control input parameters are set and the start condition is not yet evaluated.

The figure shows in alternative branches two different execution alternatives for the workflow instance, labeled by the respective states the workflow instance takes. In the

upper branch, the start condition is evaluated to false, since value 22 is transferred to that workflow instance node. Due to the defined start condition

$$sc_s(sip_i) \equiv 22 < 20 \wedge b = \text{true-signaled} \wedge c = \text{true-signaled} \equiv false$$

the workflow instance node cannot be executed. Consequently, the workflow instance enters the eliminated state; this state is graphically represented by a gray shading. In the lower part of Figure 3.12, the start condition is evaluated to true, since

$$sc_s(sip_i) \equiv 12 < 20 \wedge b = \text{true-signaled} \wedge c = \text{true-signaled} \equiv true$$

Consequently, the workflow instance is in the running state, and the workflow instance is executed. On its termination, the control parameter is set to true-signaled, and the output parameter is set to “y”. Eventually, the workflow instance reaches the done state.
As indicated above, workflow instances are created according to levels of the workflow schema tree, informally described as level-wise creation of workflow instances or shallow instantiation. As opposed to this approach, deep instantiation is another option. In deep instantiation, the complete workflow tree will be instantiated before the top-level workflow instance starts. This approach is impractical for the following reasons:

- *Long start-up phase:* The start-up time for a workflow instance will be rather long, since all workflow instances have to be created before the workflow can perform its first activity. Long start-up times for workflows are not desirable from an application-oriented point of view, since additional waiting times are introduced by the workflow management system.

- *Obsolete work:* During a particular complex workflow instance, in general not all sub-workflow instances have to be performed. This situation occurs, for instance, in the credit request workflow, where either the accept credit or the reject credit sub-workflow instance has to be performed. Assuming the general case in which workflow instance nodes are complex and therefore roots of workflow instance subtrees, creating sub-trees for all potential alternative workflow instances amounts to performing obsolete work, since for each workflow instance one of the two sub-workflow instances is required, while the other is obsolete. In a level-wise instantiation the creation of workflow instances can be deferred until the workflow management system decides which alternatives are actually required for a given complex workflow instance. More precisely, eliminated complex workflow instance nodes need not be refined by their respective workflow instance graphs.

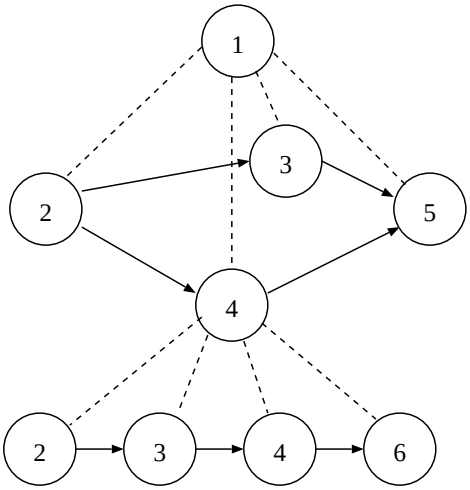

Figure 3.13: Workflow Schema with Recursion.

- *Recursive workflows:* As indicated earlier, recursion in workflow schemas is supported. This means that a workflow schema can occur as a sub-workflow schema of itself. For instance, workflow schema s can have a number of sub-workflow schemas, among which is s. In this case, a deep creation of workflow instance amounts to the creation of an infinite number of workflow instances of the recursive workflow schema s.

For an example of a recursive workflow schema, consider Figure 3.13, in which workflow schema 4 appears as a recursive sub-workflow schema. A deep instantiation of workflow instances according to workflow schema 1 will result in an infinite loop, caused by the recursion. This situation is shown in Figure 3.14, in which the ellipsis indicates the infinite creation of workflow instances resulting from the recursion.

If the workflow instances are created according to the levels of the workflow schema tree then a new workflow instance of the recursive workflow schema will only be created if and when it is actually required. If it is no longer needed then the recursion stops at that point, and the earlier recursions can be completed. This situation is shown in Figure 3.15, in which a recursion of several steps is shown. In the third workflow instance based on workflow schema 4, the workflow instance does not need to be executed: The start condition of $4'''$ is evaluated to false, and the recursion stops.

To summarize, the workflow instance is performed as follows: After initiating the first level of sub-workflows, i.e., the immediate sub-workflows of $1'$, workflow instance $2'$ is performed, followed by the concurrent execution of $3'$ and $4'$. The start condition of $4'$ evaluates to true, and the sub-workflows of $4'$ are instantiated at this point. The execution continues with sub-workflow instances $2''$, $3''$ and $4''$. The start condition of $4''$ evaluates

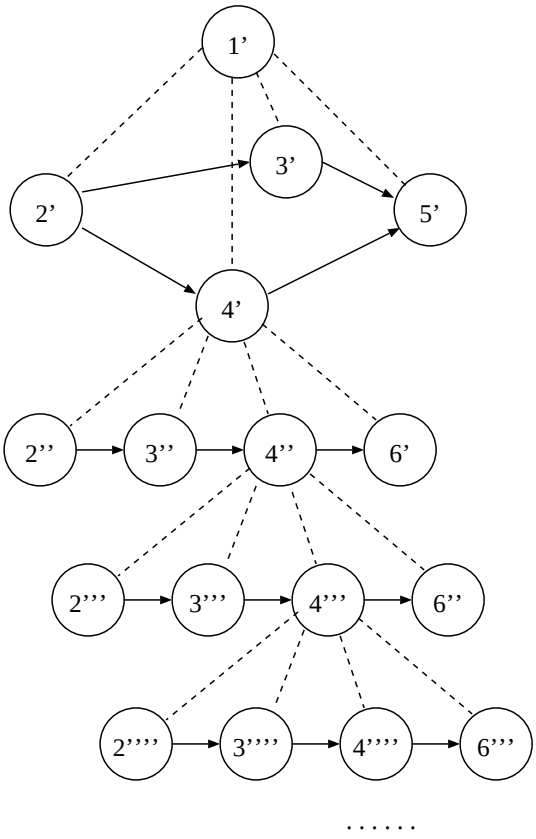

Figure 3.14: Workflow Instance with Deep Instantiation: Infinite Recursion.

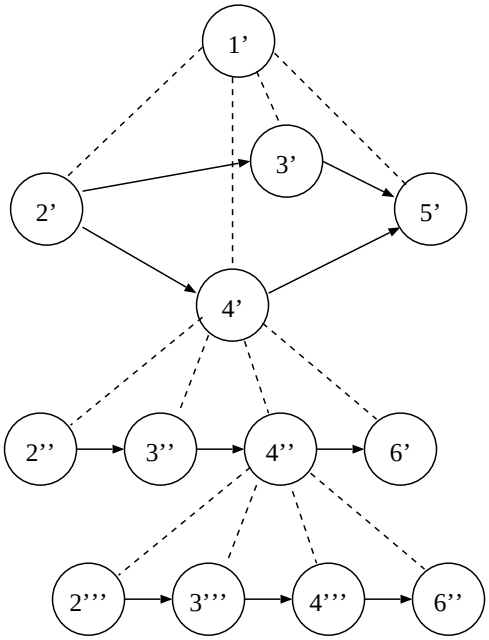

Figure 3.15: Workflow Instance with Shallow Instantiation: Finite Recursion.

to true, resulting in the next recursion, i.e., the instantiation of sub-workflows on the third level. The start condition of $4'''$ is evaluated to false, terminating the recursion. Workflow instance $6''$ is passed control flow parameter false-signaled, and terminates. At this point, the complex workflow instance $4''$ can terminate, and true-signaled is transferred to $6'$. Finally, $4'$ completes, at which time the control flow edge to $5'$ is signaled which is executed to complete complex workflow instance $1'$.

Based on the formal description of workflow schemas and workflow instances and the discussion of the dynamic behavior of workflow instances using a state transition diagram, the execution of workflow instances is now characterized in an algorithmic form.

An auxiliary function *check* is introduced to determine whether all control input parameters of a given workflow instance are set, i.e., whether the start condition of a workflow instance can be evaluated. The check function maps the set of workflow instances to the boolean values; it is defined by

$$check(i) = \text{true} :\Leftrightarrow \nu(p) \neq \perp,\ \forall p \in sip_i,\ \text{TypeOf}(p) = \text{CCType}$$

The algorithm shown in Figure 3.16 specifies the execution of a complex workflow represented by a workflow instance graph. Conceptually there is a repeat-until loop which processes the sub-workflow instance nodes of the workflow instance graph. However, sub-workflows of a complex workflow can be executed concurrently, indicated by the re-

for all $j' \in V_s$ **do**
  create $j \in V_i$ such that SchemaOf$(j) = j'$
repeatpar
  choose $j \in V_i$, such that $check(j) = true$
  if $sc_s(sip_j)$ **then**
    execute j
    set $\nu(p),\ \forall\ p \in out_j$ such that TypeOf(p) $\neq$ CCType
    $\nu(cc) :=$ true-signaled
    for all $p \in out_j$ such that $(p, q) \in D_i^h \cup D_i^v \cup C_i$ **do**
      $\nu(q) := \nu(p)$
    od
  else
    $\nu(cc) :=$ false-signaled
  fi
until all $j \in V_i$ are processed

Figure 3.16: Algorithm for Executing Complex Workflow Instance.

peatpar keyword. This means that if $check(j) = true$ for multiple sub-workflows j of the complex workflow instance then all these sub-workflows can be executed concurrently. As will be explained in Part II, the concurrent execution of sub-workflow instances is achieved by a distributed approach to workflow execution control.

A workflow instance graph i with a set V_i of sub-workflow instance nodes is executed as follows: For all $j \in V_i$ whose control input parameters are signaled, the start condition can be evaluated. Depending on the values the start conditions are evaluated to, the respective workflow instance nodes are executed. We mention that "execute i" amounts to an invocation of an external application program to implement workflow instance i, if i is an atomic workflow. If i is a complex workflow then the sub-workflows of i are initiated and executed according to the algorithm shown in Figure 3.16, i.e., the algorithm is applied recursively as governed by the structure of the workflow schema.

On the termination of a sub-workflow instance, the output parameters are set, and the values are transferred to follow-up workflow instances, as specified by data flow and control flow in the respective workflow schema graph. Passing control information allows other sub-workflow instance nodes to start. This procedure is iterated until all sub-workflow instance nodes in V_i are processed. The super-workflow instance i terminates if and when all its sub-workflow instances have terminated, i.e., if all control output pa-

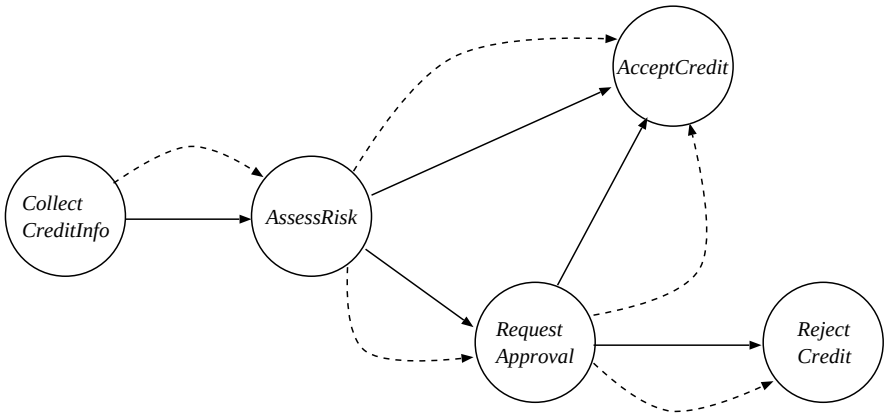

Figure 3.17: Sample Workflow Schema, Abstract Representation.

rameters are set. Hence, a complex workflow instance i based on a workflow schema s with a workflow instance graph G_i terminates if and only if all sub-workflow instances have signaled their control output parameters, i.e., if the following condition holds for all workflow instance nodes $j \in V_i$:

$$\nu(p) \neq \perp,\ \forall p \in out_j,\ \text{such that TypeOf}(p) = \text{CCType}$$

3.2.5 Sample Workflow

In order to illustrate the notations and algorithms introduced above, a sample workflow schema and corresponding workflow instances based on the credit request application process are introduced. An abstract representation of the sample workflow schema is shown in Figure 3.17, which displays the sub-workflow schema nodes and control flow and data constraints between sibling workflow schema nodes.

On the level of abstraction shown in Figure 3.17, detailed information on the parameters of workflow schemas and on the start conditions of sub-workflows is not provided. However, the figure describes the overall structure of the workflow. The complex workflow schema is composed of five sub-workflow schemas, characterizing the collection of credit information (CollectCreditInfo), the assessment of the risk of granting the credit request (AssessRisk), an additional checking activity (RequestApproval), and the final acceptance (AcceptCredit) and rejection workflow schemas (RejectCredit). The control flow and data flow edges drawn in Figure 3.17 comply with the intuitive understanding of credit request processes in banking environments.

While the abstract representation shown in Figure 3.17 is probably well suited for business persons to discuss the overall structure of this application process, it does not

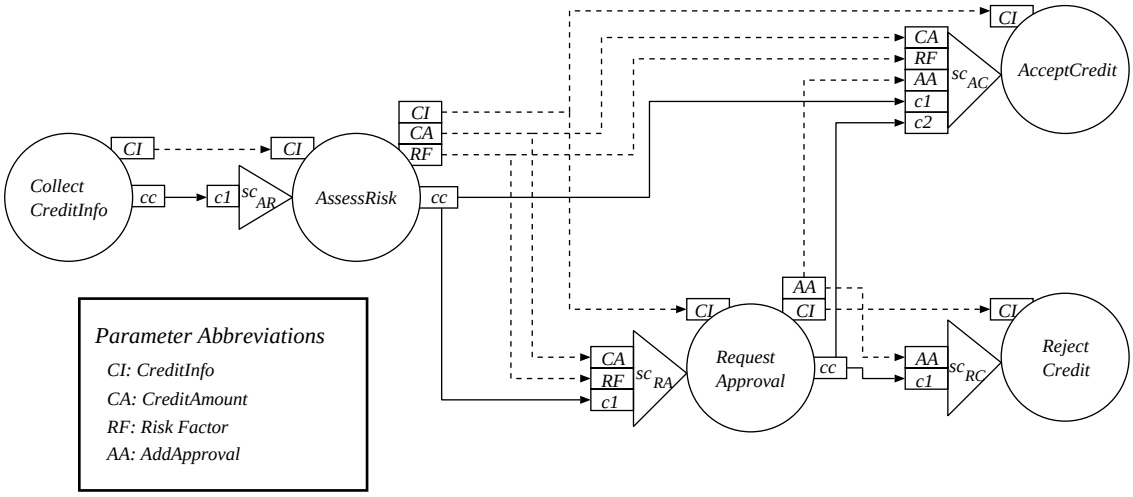

Figure 3.18: Sample Workflow Schema, Detailed Representation.

provide the detailed information required for the controlled execution of credit request application processes by a workflow management system. Therefore, Figure 3.18 displays a more detailed representation of the sample workflow schema, including control parameters and data parameters of workflow schema nodes, data and control flow edges as well as a structural representation of the start conditions of the sub-workflows involved; the start conditions are given in the text. Starting on the left hand side of the workflow schema, the start condition of the CollectCreditInfo workflow schema is the boolean constant true, indicating that all credit request workflows start with a CollectCreditInfo sub-workflow. The other start conditions are defined as follows:

- The start condition of the AssessRisk workflow schema is given by

  $$sc_{AR} \equiv c1 = \text{true-signaled}$$

  Hence, the start condition of the AssessRisk workflow schema can be evaluated if and when the CollectCreditInfo workflow is terminated, which is signaled by setting the control output parameter to true-signaled. This value is transferred to start input parameter $c1$ of sc_{AR}. Then the start condition is evaluated to true, and the AssessRisk workflow can be started. For a more concise presentation, we use c to indicate c = true-signaled in start conditions for a start input parameter c representing control flow.

- The start condition sc_{RA} of the RequestApproval workflow schema can be evaluated after true-signaled is received in control input parameter $c1$. Now the control flow

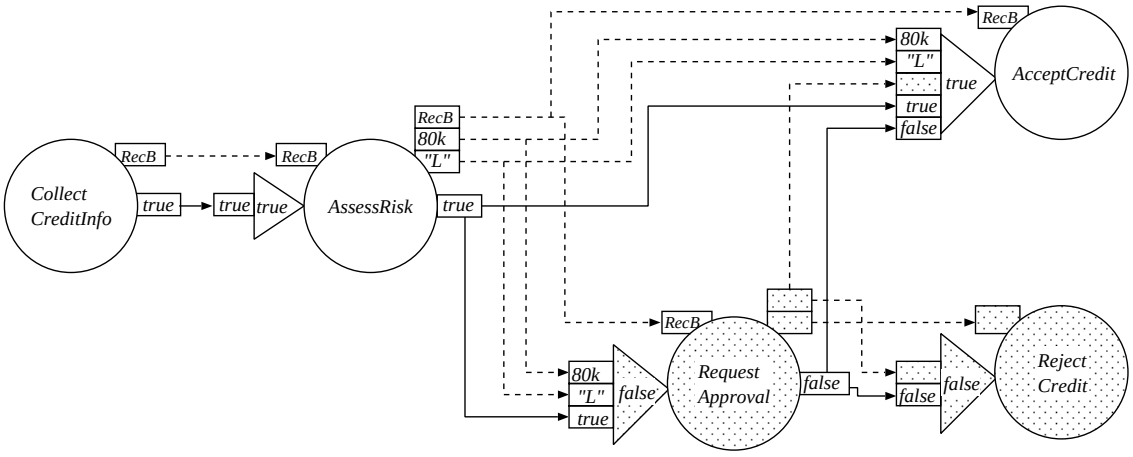

Figure 3.19: Sample Workflow Schema: Credit Accepted.

relevant data flow as specified by input parameters CA and RF, respectively, is used to evaluate the start condition:

$$sc_{RA} \equiv c1 \wedge (CA \geq 100.000 \vee RF = "H")$$

If this condition evaluates to true then the RequestApproval workflow is started, the output parameter AA is set to either "Y" or "N", indicating a granting or a rejection of the credit request, respectively.

- sc_{AC} can be evaluated after signaling is received from the control flow parameters of the AssessRisk and RequestApproval workflows in $c1$ and $c2$, respectively. In this case, the start condition

$$sc_{AC} \equiv (c1 \wedge (CA < 100.000 \wedge RF = "L")) \vee (c2 \wedge AA = "Y")$$

is evaluated.

- Finally, the start condition of the RejectCredit workflow is evaluated, which is done after receiving the signal from RequestApproval in $c1$. The start condition is

$$sc_{RC} \equiv c1 \wedge (AA = "N")$$

which means that the credit request is finally rejected if the RequestApproval workflow decides so, i.e., if its output parameter contains the string "N".

Given this detailed specification of the sample workflow including start conditions of sub-workflows, particular workflow instances based on that workflow schema are discussed. The first workflow instance to be investigated is shown in Figure 3.19. The

execution of that workflow instance is based on the algorithms for executing workflows as presented above. Assume the complex workflow has started and its sub-workflow instances have already been created. According to the algorithm shown in Figure 3.16, a sub-workflow instance i is determined for which $check(i) = true$ holds. This is the case for the CollectCreditInfo sub-workflow instance. (For ease of convenience we mark the sub-workflow instances by their workflow schema name, bearing in mind that these are workflow instances and that the different examples discussed here refer to different workflow instances.) Since its start condition is true, it is executed. On its termination it writes its output parameter RecB, representing the credit information of a customer B. Since the data structure of this record is irrelevant in this example, it is omitted. A true-signaled value in its control output parameter indicates the execution of that workflow instance. In the next step, the values of its output parameters are transferred to sibling sub-workflows, as specified in the workflow schema. In particular, the true-signaled value of the control output parameter is transferred to the control input parameter of the AssessRisk sub-workflow, while RecB is transferred to the input parameter of that sub-workflow.

At this point, the AssessRisk sub-workflow instance is ready for execution, since $check(\text{AssessRisk}) = true$. In this particular workflow instance, the credit amount is 80k, an abbreviation of 80.000 Euro, introduced due to space limitations in the graphical representation. The credit information is stored in RecB, which was generated during the execution of the CollectCreditInfo sub-workflow. AssessRisk takes the information stored in the credit information record and assesses the risk of granting that credit request. The risk is assessed as low, indicated by the output parameter value "L". The other application-specific output parameters represent the credit information record and the credit amount requested, respectively. The credit amount is retrieved from the credit information record to be transferred to the start conditions of the follow-up sub-workflows. Now the parameter values are transferred to the RequestApproval and AcceptCredit workflows. The RequestApproval workflow cannot be executed since its start condition sc_{RA} is evaluated to false: setting $\nu(c1) = true$, $\nu(CA) = 80.000$ and $\nu(RF) = "L"$ yields the expression

$$sc_{RA} \equiv true \wedge (80.000 \geq 100.000 \vee "L" = "H") \equiv false$$

Hence, the RequestApproval workflow is not executed. The control flow output parameter of that workflow is set to false-signaled, which is transferred to the start conditions of the AcceptCredit and RejectCredit workflows.

Since by now all control flow input parameter are signaled for the AcceptCredit workflow instance, its start condition can be evaluated. Filling the parameter values in the start condition sc_{AC} yields the expression

$$sc_{AC} \equiv (true \wedge (80.000 \leq 100.000 \wedge "L" = "L")) \vee (false \wedge \bot = "Y") \equiv true$$

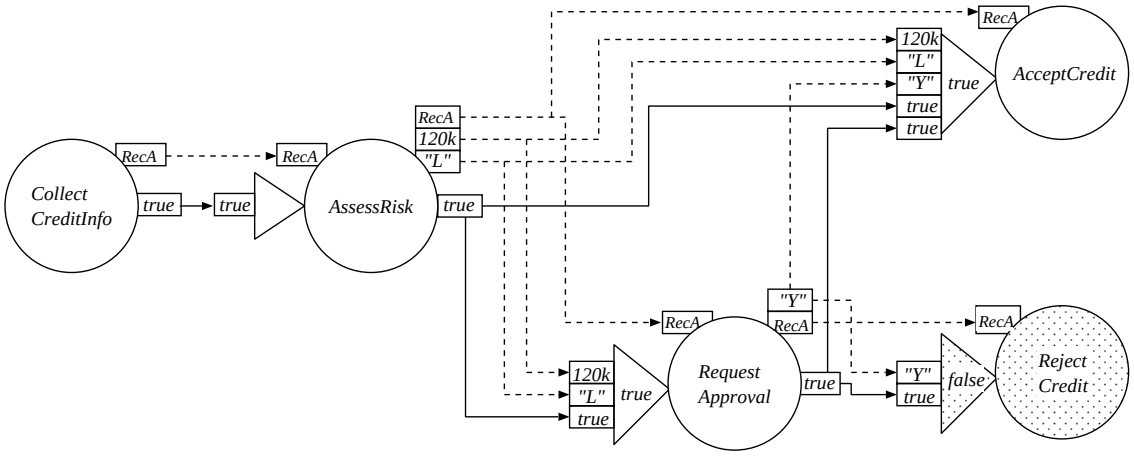

Figure 3.20: Sample Workflow Instance; Credit Accepted after Approval.

Hence, the credit is accepted, and the AcceptCredit sub-workflow instance is executed. Notice that comparisons with undefined parameters (represented by the $\perp$ value) are permitted, yielding the false value. Obviously the RejectCredit workflow cannot be executed since its start condition evaluates to false. By now all sub-workflows of the complex workflow instance are processed, indicating the termination of the top-level workflow instance.

Figure 3.20 shows a different workflow instance based on the credit request workflow schema. In this example, the credit amount is 120.000 Euro, and the AssessRisk workflow assigns a low risk level for that credit request, indicated by $\nu(RF) = "L"$. Hence, the start condition sc_{RA} of the RequestApproval sub-workflow instance is provided parameter values $\nu(c1) = true$, $\nu(CA) = 120.000$ and $\nu(RF) = "L"$, which yields the expression

$$sc_{RA} \equiv true \wedge (120.000 \geq 100.000 \vee "L" = "H") \equiv true$$

The RequestApproval workflow instance takes the decision to accept the credit and fills the AddApproval output parameter accordingly with "Y". Since the sub-workflow instance was executed, its control output parameter is set to true-signaled. By now all control flow input parameter are signaled for the AcceptCredit workflow instance, and its start condition can be evaluated. Providing the start condition with the actual parameter values yields the expression

$$sc_{AC} \equiv (true \wedge (120.000 \leq 100.000 \wedge "H" = "L")) \vee (true \wedge "Y" = "Y") \equiv true$$

Hence the AcceptCredit workflow instance is executed, and the credit request is finally accepted.

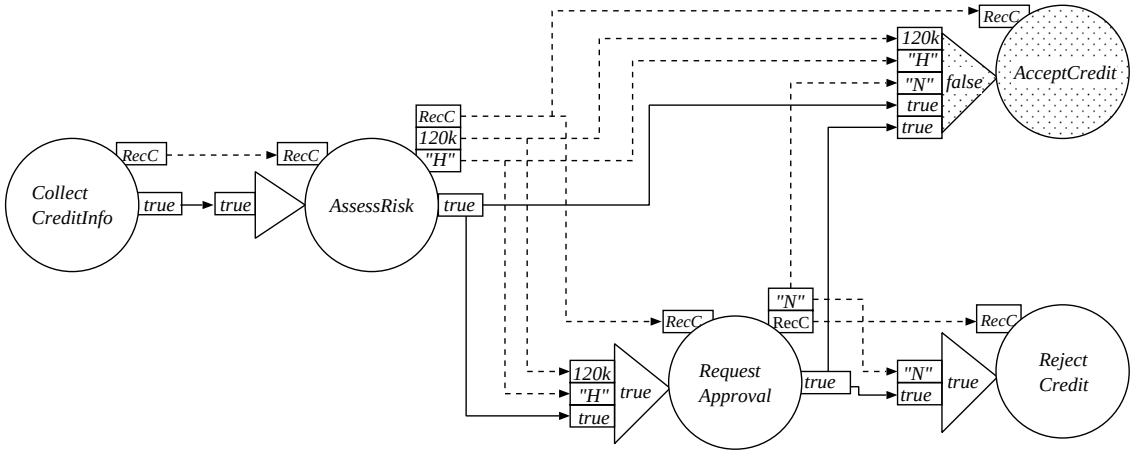

Figure 3.21: Sample Workflow Instance; Credit Rejected.

In the final workflow instance discussed, the credit request is rejected; it is shown in Figure 3.21. The credit amount is 120.000 Euro, and the AssessRisk workflow assigns a high risk level for that credit request. Hence, the start condition sc_{RA} of the RequestApproval sub-workflow instance is provided parameter values $\nu(c1) = true$, $\nu(CA) = 120.000$ and $\nu(RF) = "H"$, which yields the expression

$$sc_{RA} \equiv true \wedge (120.000 \geq 100.000 \vee "H" = "H") \equiv true$$

The RequestApproval workflow instance takes the decision not to accept the credit and fills the AddApproval output parameter accordingly with value ”N”. Since it is executed, the control flow output parameter is set to true-signaled.

3.3 Summary

This chapter investigates workflow specifications. In the first section, a graph-based workflow language is introduced informally, guided by workflow aspects. This workflow language will be used throughout this thesis to specify workflows. Section 3.2 provides a formalization of workflow schemas and workflow instances. While the correctness of workflow schemas it highly application-specific, it has to be defined by the workflow modeler. However, consistency criteria are defined which have to be satisfied by workflow schemas. The presentation of workflow instances and of the semantics of their executions is based on a state transition diagram. An algorithm that defines how workflow instance graphs are executed is introduced. The notations and algorithms are illustrated by a workflow schema and a number of workflow instances based on that schema.

Chapter 4

Workflow Application Development Processes

While a formal foundation of workflow schemas and workflow instances is an important basis for research in workflow management and for developing robust and efficient workflow management systems, this chapter takes a broader look at workflow management. In particular, it investigates workflow application development processes as specific software processes. Naturally, techniques and methods from the software engineering domain are investigated and adapted to the specific requirements of the development of workflow applications. In so far, the technical aspects in workflow management are put aside for a moment, and the way workflow applications can be developed is in the center of this final chapter of Part I.

Software development processes have been investigated for some time now [8, 75]. However, the specific properties of workflow application development processes received little attention so far in the software engineering community. This chapter tries to remedy this situation by analyzing problems encountered during the conduction of real world workflow projects and by formalizing the experiences drawn as a workflow application development methodology, which can be adapted to the needs of particular workflow projects. While the general structure of the methodology is based on techniques known from software engineering process models, the specific properties of workflow applications and their implications to development processes of workflow applications are well taken care of.

While workflow application development processes differ from one project to the next, their general structure can be described as follows [50]: The first phase of workflow application development processes typically deals with gathering information, relevant in the application domain. In this phase typically empirical studies based on interview tech-

niques and analysis of available documentation are conducted. It is important to notice that the activities in this phase are centered around the application domain, rather than the technical issues involved. The next phase involves business process modeling, in which the information gathered is used to specify business process models. The main purpose of business process modeling is to provide a general and easy-to-read notation of business processes, which enables information system experts and domain experts to validate and optimize business process models. The result of this phase is specified in a business process model, which is used as a basis for the next phase, the workflow modeling phase. The aim of the workflow modeling phase is to enhance the business process model with information needed for the controlled execution of workflows by a workflow management system, involving adding technical information and purging application-specific information which is irrelevant for the purpose of workflow management. In this phase, workflow languages are used. Finally the workflow application is implemented using a suitable workflow management system, it is deployed in the target environment, and the operational phase starts.

While this overview on workflow application development processes gives a coarse idea on the general structure of these processes, it is fairly incomplete. For instance, the linear structure implied by the discussion is not realistic. In contrast, a typical pattern (not only) in workflow projects is that during a phase the project participants find out that the information generated in the previous phase is incomplete. In this case the previous phase is re-executed to gather the missing information and to be able to proceed with a sufficiently complete and correct information basis. This pattern is known as incremental improvement, and it is widely used in software engineering processes.

4.1 Motivation and Issues

In this section, workflow projects are sketched, and typical issues related to the planning and conduction of workflow projects are investigated. For more detailed information, the reader is referred to [110]. The workflow application development methodology is based on lessons learned from the conduction of a number of workflow projects. The key properties of these projects are described in [35].

When examining the different development processes and the problems encountered in the respective workflow projects, two classes of relationships can be identified between problems emerged during workflow projects: The first class combines those cases, where the problem can directly be shown by the existence or by the causal order of development activities (Problems 1 to 3, see below). The second class consists of cases, in which spe-

cific problems occurred during one particular activity (Problems 4 to 6). This distinction is important for the treatment of the problems by the workflow application development methodology, as will be discussed below.

> *Problem 1*: The organizational and technical aspects of the workflow schema have often been worked out independently.

Organizational and technical aspects depend on each other strongly, because there are technical constraints (like the integration of legacy systems) which have a strong influence on the design of organizational structures. An important flavor of this problem is the modularity of legacy applications. Often these applications have a much coarser granularity than required by the workflow application. These implications of existing information systems have to be taken into account in an early project stage. In other words, the technical solutions as implemented in the workflow application should always be derived from organizational requirements. For this reason, the organizational and technical aspects should be designed closely related in an iterative way.

> *Problem 2*: In most cases, the selection of a workflow management system was done in a very early project stage.

The workflow projects examined spent different amounts of time and resources in selecting a workflow management system. Interestingly enough, the selection of a suitable workflow management system was among the earliest phases in the majority of the analyzed workflow projects. Two of the examined projects set up an extensive workflow management system selection process, whereas other projects used an existing working relationship to a specific vendor or other non-technical arguments for their decision. In most projects, the early selection has been identified as one of the main problems, since the selected workflow management system could not cope with the specific requirements of the respective workflow project. If the deficiency of the selected workflow management system was detected in a late project stage (often only in the implementation phase of the project), specific functionality had to be coded in order to fulfill the needs required by the workflow application. The selection of an unsuitable workflow management system caused major adaptations in later project stages, which generated considerable overhead in time and resources. We remark that the selection of a workflow management system is not a trivial task, since there is no defined standard for workflow management systems, their supported functionality, user interface issues and even their software architecture or interfaces to legacy applications. This fact leads to the problem, that different workflow management systems fit different application requirements. As a result, a survey phase

including the gathering of technical and organizational information should be completed before selecting a workflow management system.

Problem 3: The development process has been done without prototyping.

Only in one project, a prototype of the target application was developed. In most cases, prototyping was either not considered necessary for developing the workflow application or it was omitted due to project delays. It turned out, however, that prototyping is a very helpful activity in building workflow applications. The main reasons are as follows: Firstly, the organizational requirements of the business process can be validated in prototypes by the users of the system. It turned out that system usability and front-end design are important factors for the acceptance of the new technology by the workflow participants. Gaining support from the employees, i.e., the users of the workflow application, is an important success factor for the workflow project, and a suitable and functional prototype can foster the acceptance of the workflow application. Secondly, using a prototype, technical restrictions can be tested in an early project stage. Prototyping allows to detect technical deficiencies of the workflow management system selected or of the technical infrastructure of the organization — it is good to know these problems in early project stages in order to be able to react timely to these issues. Prototyping can help (i) to enhance acceptance of the new technology by the workflow participants, (ii) to detect upcoming technical problems in early project stages and (iii) to decide whether workflow is the right technology to support the application processes.

Problem 4: An automatic transfer of business process models into workflow models proved to be unsuitable.

If business process modeling and workflow modeling were carried out with different tools, an automatic transfer of the business models into workflow models led to unacceptable results. We believe that a fully automatic transformation of business process models to workflow models is not feasible since they focus different aspects. In particular, business process modeling treats in detail the demands of the application from an application-oriented point of view, whereas workflow modeling focuses on the technological aspects of the application process and its organizational and technical environment. As a result, these domains are typically covered by different people using different terminology, i.e., business terminology versus workflow terminology. In the technological side, there are no suitable standards for the transformation of business process models in workflow schemas. For example, the Workflow Management Coalition standard specifies very few low-level object types, e.g., activity [118]. As a result, the translation of a business process model to a workflow schema remains imprecise, and the expenditure on subsequent

modifications of the workflow schema can be expected to be very high. However, the manual transformation of business process models into workflow schemas proved to be difficult, too. This is mainly due to the different expressive powers of business process modeling languages and workflow languages.

> *Problem 5*: The integration of legacy systems was a critical factor for the success of workflow projects.

In all case studies, the integration of complex legacy systems in the workflow application was necessary. In none of the examined projects only simple applications like office applications had to be integrated. Besides, in all cases application integration was considered to be a critical success factor of the project, since a re-implementation of legacy applications was not an option. Hence, considerable effort was necessary to integrate the different legacy systems into the workflow application. One of the main issues in this context was the development of interfaces between the workflow management system and the external legacy applications. In some cases where the characteristics of the legacy systems were identified at a late project stage, the development of proprietary code was required for the integration. Additional problems like poor performance or lack of robustness of the workflow application were sometimes created by unsatisfactory integrations. Wrapper techniques are used to wrap the specific properties of legacy applications and to present a uniform interface to workflow management systems. In this context, object-oriented frameworks like the OMG Business Object Facility [21, 13] or IBM's San Francisco Framework [9, 10] are promising approaches; these will be discussed in Part III.

> *Problem 6*: Severe performance problems could be identified during the field test phase.

During field test, every examined workflow project had considerable problems regarding the performance of the workflow management application, if the application had to cope with a large number of users and workflow cases. In one case the application was lacking reliability, too. Although the integration of the legacy systems could be identified as one cause for these deficiencies, the workflow management systems to a large extent are responsible for the technical problems. In another case, the underlying database used by the workflow management system proved to be unsuitable for the handling of large amounts of data. Apart from the fact that the database size was limited, the performance of the database led to a response time of more than three minutes for starting workflow instances. During this time, the user hat to wait, definitively an unacceptable situation.

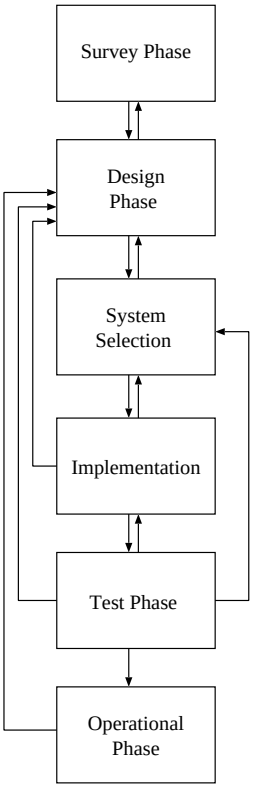

Figure 4.1: Workflow Application Development Methodology.

4.2 Workflow Application Development Methodology

This section presents a workflow application development methodology, based on the notion of a phase. A phase defines a time interval during which particular activities are performed. Typically, phases consist of related sub-phases or activities, which are carried out by a group of persons, using a set of input documents and developing a set of output documents. Based on a high-level description of the methodology, the internal structure of the phases and the relationships between phases are shown in some detail.

4.2.1 Overview

Rather than presenting a formal method for describing development process models, an informal notation is used, in which phases are represented by boxes, and information or causal constraints between phases are described by directed arcs between boxes. Phases can be nested, i.e., each phase can consist of a number of sub-phases. Sub-phases are also called development activities or simply activities. Despite of being rather informal, the methodology hints workflow project planners and managers to plan and conduct workflow

development projects.

The overall workflow application development methodology is shown in Figure 4.1; its phases are sketched as follows: The first phase of the workflow application development methodology is the Survey phase. In this phase, the goals of the project are defined, the project team is established, and initial information on the application domain is gathered. The main result of the Survey phase is a reviewed as-is business process model. The Design phase is next, in which the developed business process model is analyzed and optimized to reflect the overall goals of the business, specified as a to-be business process model. (We remark that business process modeling and re-engineering techniques are outside the scope of this thesis.) Based on the as-is business process model, the project management decides whether workflow technology is adequate to support the requirements of the business process under investigation. If so, a suitable workflow management system is selected based on the requirements specified in the to-be business process model, and the Implementation phase starts. The Test phase is next, which includes lab test and field test, as refined below. If the tests are successful, the Operational phase is reached.

We remark that performing these phases sequentially represents an ideal situation, which most likely will not be appropriate in most workflow projects. To capture typical exceptions in the development process, we introduce additional edges, e.g., certain phases or sub-phases can be re-performed during the development process. Defining and controlling the conditions under which development phases can be iterated is an important task of the project management, and it can be assisted with the workflow application development methodology.

We denote the property to step back in previously conducted phases as *evolutionary*, since development steps may iterate, incrementally improving the artifacts, for instance business process models and workflow schemas. In doing so, a high degree of flexibility in developing workflow applications is provided. By defining possible iteration structures, the methodology helps in ruling out jumps that are not meaningful. We mention that the evolutionary structure is present both within phases and between phases, as is elaborated below.

4.2.2 Survey Phase

The overall goals of the Survey phase are twofold. First, initial information on the domain is gathered to decide which business processes should be supported. The second goal of this phase is to develop a reviewed as-is business process model of the selected processes which contains both organizational and technical information; the detailed structure of

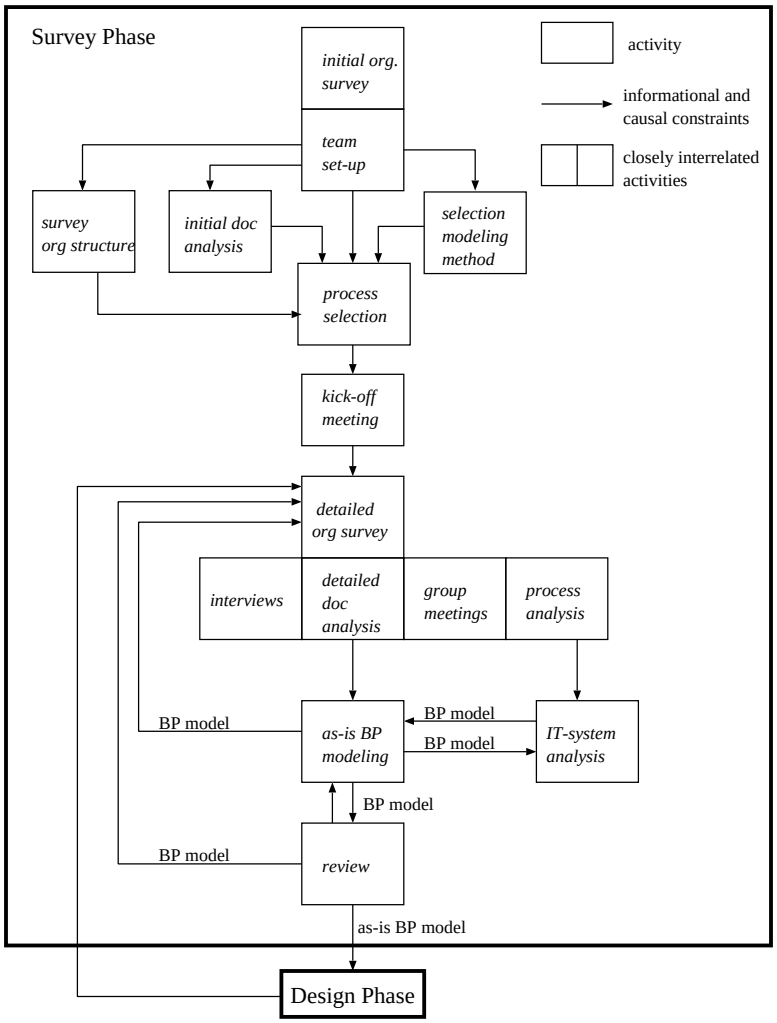

Figure 4.2: Survey Phase.

the Survey phase is shown in Figure 4.2. Technical issues should also be taken into account in this early phase in order to identify possible restrictions due to limitations of the information systems to be integrated.

The Survey phase starts with an initial survey of the business process on an abstract level in order to get an overview of the application process and the persons and organizational units involved. Besides, the team set-up for the Survey phase is done based on this information by identifying the persons to participate in the detailed survey. The next activities comprise a closer look at both the organizational structure of the organization and the documents that describe the application domain, including legislative regulations and policies. This information is helpful for the process selection step, in which the relevant application processes are selected for the workflow application. In addition, application-

specific information is useful for the preparation of the survey team kick-off meeting. In this meeting, the overall goals of the project, the characteristics of the approach used, the application processes to support, and the next project activities are presented to the team members. A strong participation of the employees involved in a workflow project is an important factor for the quality of the business process model and, eventually, for the quality and acceptance of the workflow application.

After the modeling method for the business process model is selected, the main activity of the Survey phase, the detailed organizational survey, is carried out. There are several options to obtain the relevant information in this sub-phase. Interviews with survey team members are a costly but effective method to get detailed knowledge on the activities within the selected business processes. We stress that a detailed analysis of the documents is suitable in most cases. The detailed survey yields an initial version of the as-is business process model. If information is missing or ambiguous then the corresponding parts of the detailed organizational survey activity have to be repeated. In any case, the technical infrastructure of the organization is analyzed. The business process model is then extended to this aspects.

When a complete version of the as-is business process model is created, a review meeting including the entire survey team and the project management is carried out. The business process model is presented to the team, and it is discussed with the team in order to eliminate potential inconsistencies in the model. Either these inconsistencies can be clarified in the meeting or the survey process iterates, and a new version of the business process model is developed after another organizational survey iteration. The new business process model is again discussed in a review workshop. If all team members agree on the as-is business process model, the Survey phase is completed.

4.2.3 Design Phase

The Design phase aims at analyzing and optimizing the as-is business process model; the internal structure of this phase is shown in Figure 4.3. The as-is business process model serves as an input for this phase. The first set of activities deal with organizational and technical modeling of the so called to-be business process model, which represents the optimized business process which will be supported with the workflow application. The part involving the design of the organizational aspects can be subdivided into three sub-activities: process modeling, organizational structure modeling, and data modeling. Beside organizational properties, technical properties of the application environment, like the computational infrastructure, available applications, and the overall architecture should be reflected in the to-be business process model.

Organizational and technical to-be-modeling should be carried out in an evolutionary way, so that the strong dependencies between technical and organizational aspects can be taken into account and modeled explicitly in the to-be business process model. At this point, fundamental knowledge on the business process is available. Using this knowledge on the process and on the organizational and technical environment in which it is performed, the team decides if workflow is an adequate technology to support the business process. This is done during the technology / workflow method selection activity. If so, an appropriate workflow language is selected, and the application development process continues as described in the methodology. If it is decided that workflow is not adequate then the information gathered so far can be used as valuable input for a traditional software development project.

A second set of activities during this phase are combined into to-be workflow modeling. We have already described the distinction between business process models and workflow schemas. The to-be workflow modeling activity is subdivided into workflow process modeling, organizational modeling, and data modeling. The output of this activity is a workflow schema which reflects the contents of the to-be business process model, enhanced with technical features. Since the workflow management system is not yet selected in this phase, a method for building the workflow model must be chosen independently from a particular workflow management system. As discussed above, an automatic transfer of business models into workflow models is not suitable in most cases, so appropriate resources should be provided for the workflow modeling activity.

After the workflow modeling step has generated a workflow schema, a review activity can be carried out. Changes or new requirements may concern the business process model or the workflow schema, making it necessary that both business process modeling and workflow modeling can follow this review. The review activity may show that not enough information was gathered during the Survey phase. In this case, the workflow application process may re-perform parts of the Survey phase. In particular, the detailed organizational survey activity is entered again to get the missing information, as shown in Figure 4.2. When the Design phase is completed, detailed information on the business process, its technical and organizational environment and on the workflow schema implementing the business process is present, which is valuable input for the next phases of the methodology involving the selection of a suitable workflow management system and the implementation of the workflow application.

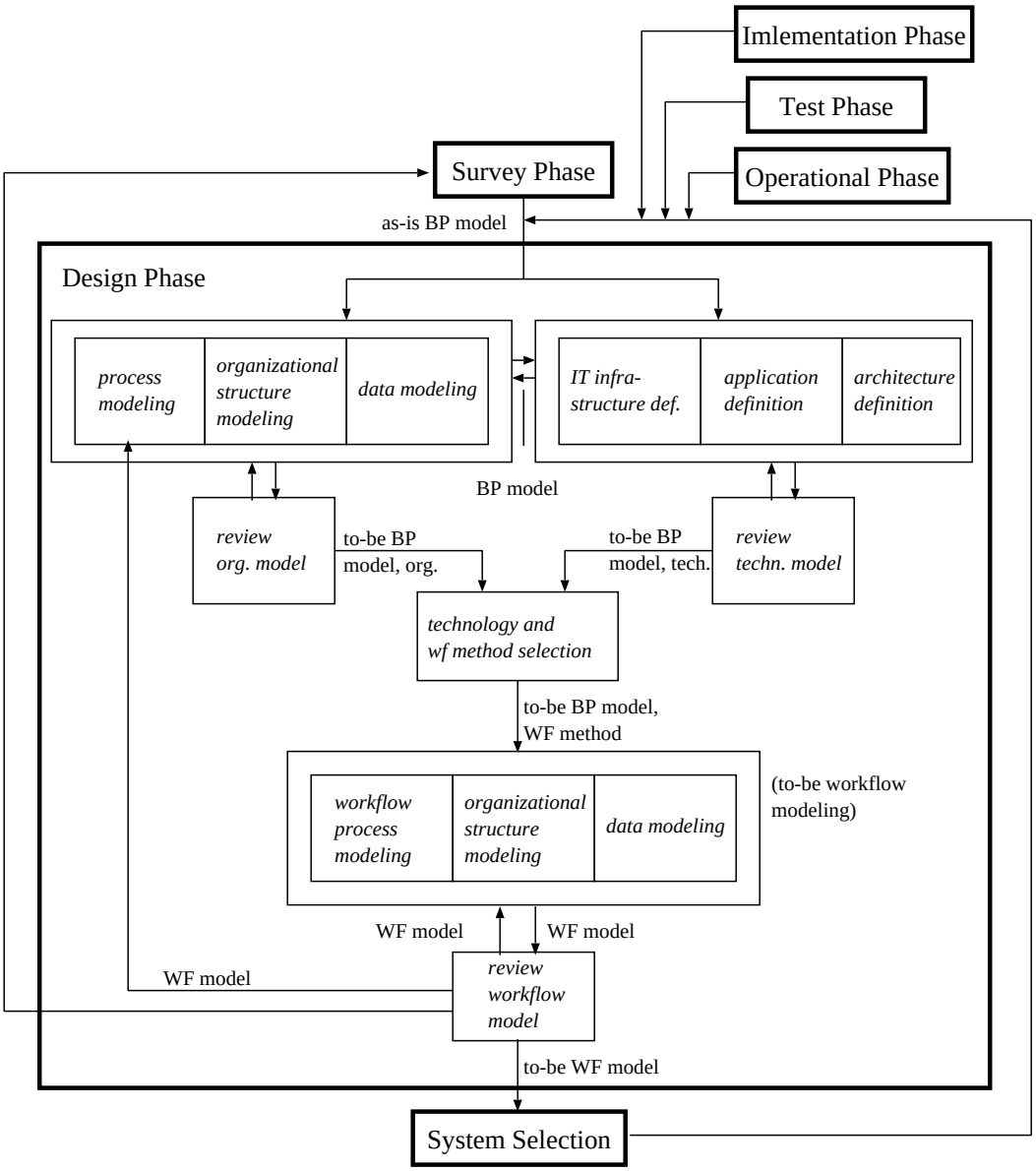

Figure 4.3: Design Phase.

4.2.4 System Selection and Implementation Phases

During the Design phase, the to-be business process model is developed, as is a representation of the workflow independent from a particular workflow management system. The main purpose of the System Selection phase is selecting a workflow management system, which is appropriate for the workflow application under development. The internal structures of the System Selection and Implementation phases are shown in Figure 4.4.

As was discussed in above, one of the main reasons for failing workflow projects is due to selecting an inadequate workflow management system. Among the potential reasons for this situation is selecting the system in an early project stage, when little or no information on the business process and on its technical requirements may be available. Consequently, it is not possible to choose a workflow management system according to the specific needs of the workflow application. However, putting effort in selecting a suitable system is important, since the workflow management systems commercially available today differ considerably with respect to their functionality, interfaces, and their ability to integrate legacy applications.

The selection process starts with defining criteria for selecting an appropriate workflow management system based on the business process model as specified in the Design phase. Obviously, there is a myriad of criteria for selecting a suitable workflow management system, some of which are discussed now. Based on studies [20, 23], the following classification of criteria is relevant in the System Selection phase:

- *Integration criteria:* Integration criteria specify application and data integration aspects. In particular, the data structures and types of application systems which can be integrated in the workflow application are taken into account. We remark that these are important criteria, since the success of a workflow project may rely on the integration of existing domain-specific applications, which typically have been developed independently from workflow applications.

- *Interaction criteria:* Interaction criteria can best be described by a set of questions, which have to be answered to select a workflow management system:

  Is the user interface of the workflow management system adequate for the users and the tasks to be supported? Does the workflow client application have adequate notification mechanisms, like push- (the system actively notifies the client) and pull-communication (the client retrieves next workflow activities from the system)? Finally, is the amount of training required for users to work with the new system appropriate?

- *Development criteria:* These criteria include the expressiveness of the workflow language underlying the workflow management system, e.g., is it possible to express control flow, data flow and alternative execution paths, and other required properties? In addition, are there powerful simulation and test facilities present? Does it support flexible support for maintenance procedures and security and integrity constraints that are required by the application? Does the system support the integration of software components and business objects in a convenient way?

- *Run-time criteria:* Does the system provide adequate functionality for the end-user as well as modeling and monitoring capabilities for process designers and for process managers? In addition, is the workflow system able to process the load expected for the application, i.e., does the system scale? Does the system save intermediate workflow states in persistent storage, so that workflows which are interrupted can be recovered safely? Are flexible workflows supported, i.e., the controlled dynamic adaptation of running workflow instances?

- *General criteria:* Is the system available for the platform that is already installed in the organization? General criteria also include the reputation and product strategy of the vendor (will there be support for the produce in five years from now?) and reference installations as well as the availability and pricing of the product.

Based on these criteria and on market analysis data, an initial set of workflow management systems can be established. However, if the systems available do not meet the criteria (or the systems cannot be used in the particular setting) then the selection criteria have to be re-defined. When a system is found that satisfies the criteria, it is tested against the criteria. When the system tests are successful then a review meeting is carried out in which the final decision on the system to be used is taken. If the requirements of the application are available before the Design phase is completed, the System Selection phase can start before business modeling is completed. This usually saves considerable amounts of time, especially if the systems to be tested require time to get hold of, to install, and to test.

The Implementation phase is based on the to-be workflow schema as specified in the Design phase and the workflow management system selected. Broadly speaking, this phase contains two major activities, one of which deals with the implementation of the workflow model according to the formalisms and rules provided by the selected system. As is shown in Figure 4.4, this activity deals with specifying process models, data models and organizational models. The second activity in this phase is concerned with tool integration, i.e., the integration of external applications.

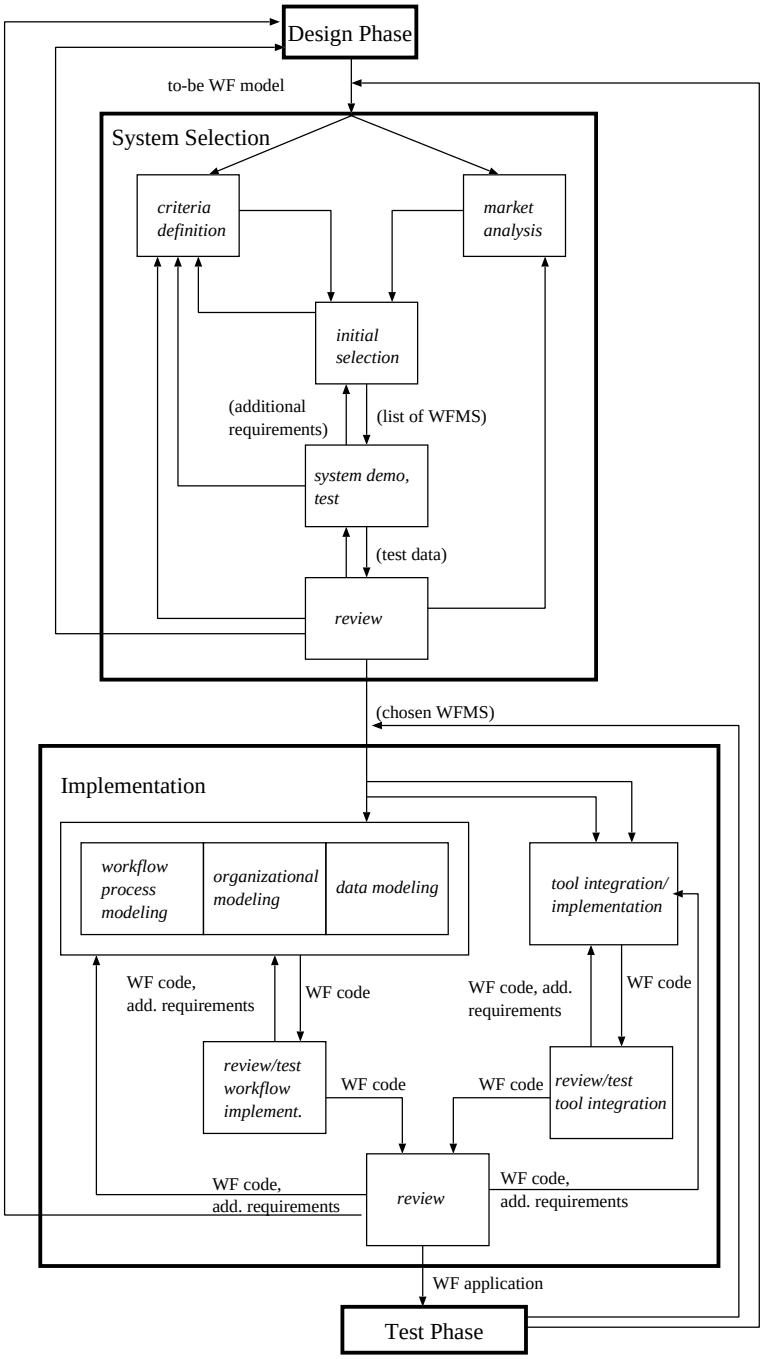

Figure 4.4: System Selection and Implementation.

Depending on the support provided by the selected system, tool integration may require considerable amount of coding and extensive testing. In case of problems, the respective activities are re-performed. This process continues until the workflow implementation and the tool integration meet the requirements imposed by the workflow schema. Finally, a review combines the results from the two activities. As specified in the figure, this process may iterate. This phase results in a workflow application, which is used in the Test phase. However, the review may result in the decision to re-perform the Design phase. In this case, the Design phase is re-entered, as shown in Figure 4.3.

4.2.5 Test Phase

The Test phase comprises the two sub-phases lab simulation and field test. Overall goal of the Test phase is to obtain information about the technical stability and the usability of the workflow application in the target environment. The details of the Test phase are shown in Figure 4.5.

During the first activity in the lab simulation sub-phase, the simulation goals are defined. This activity directly depends on the to-be workflow schema developed in the Design phase. The second task defines the test scenario. The test scenario defines the business processes and workflows to test. This includes definition of the amount of data and workflows as well as the relevant business tasks; time restrictions are also specified in the test scenario. Now, test routines have to be developed, the lab simulation is conducted, and its results are analyzed. If the simulation analysis indicates that further simulation goals are of interest, another iteration of this sub-phase is started. Re-definitions of the test scenario may also be required. If the lab simulation shows that the implemented workflow application is not suitable for the given business application, a re-design of the to-be workflow schema is necessary, as is a re-implementation of the workflow application. If on the other hand the lab simulation finishes successfully, a field test can start, which is discussed next.

A field test is performed to show that the workflow application is able to handle real world situations, characterized by problems which (at least partially) cannot be planned or predicted beforehand in laboratory environments. Therefore, the application is tested against real world conditions. After defining the goals of the field test, the business processes to be tested are selected. For each such process, a backup solution must be provided to cope with potential error situations in the workflow application. Meanwhile, the employees involved in the tested processes are trained on the new workflow application. If the training is completed and the backup solution is tested extensively and is considered stable, the field test can be performed. After its completion, the test data generated will

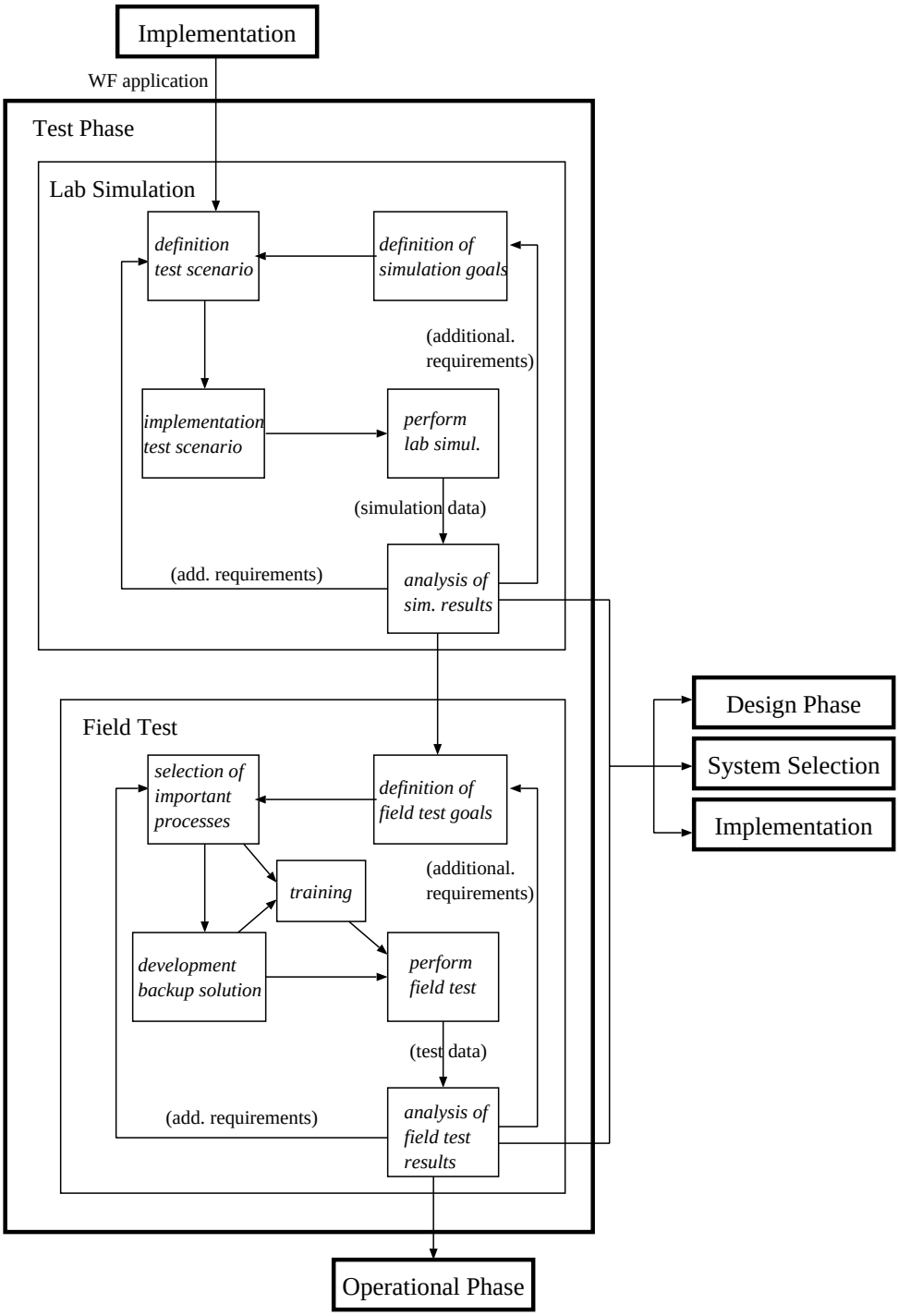

Figure 4.5: Test Phase.

be analyzed, which may even result in the definition of new test goals or which may show new requirements influencing the selection of critical processes. This process can iterate, such that the field tests become increasingly accurate.

We remark that the field test may even show that the to-be business process model or workflow schema is not suited. In this case, the respective phases, i.e., Design and Implementation, are entered again. The field test may also show that the workflow management system selected is not able to handle the workflow application, in which case the System Selection phase is re-performed. Typically, a demo license of a workflow management system is used until the Operational phase begins, to limit cost if the field test results in a change of the selected system. The review activities in this phase may show severe problems, resulting in stepping back into the Design, System Selection, or Implementation phase, as shown in Figures 4.3 and 4.4.

4.2.6 Operational Phase

The Operational phase comprises the sub-phases Installation and Run-Time as well as a set-up activity, in which the technical environment for the deployment of the workflow application is provided; the internal structure of the Operational phase is shown in Figure 4.6.

The installation sub-phase includes data migration and user training activities and the system deployment in the target environment of the workflow application. User training involves educating employees on using the new workflow application. Data migration deals with relocating data from original systems into the new workflow application or in other systems, as required. If the training and the data migration are finished successfully, the new workflow application can be deployed in the organization.

The Run-time sub-phase is characterized by performing the daily business of the organization using the new workflow application. Workflows are monitored, and execution data is gathered, which is important for the continuous improvement of the workflow application and the respective business processes. In particular, the performance monitoring generates data for the process controlling activity, which in turn produces information for a continuous process improvement task. The latter task also gets information from the operational business directly, mostly information of informal character. We remark that continuous process improvement is a complex activity, which is in the domain of business experts; hence, it is not covered in this thesis.

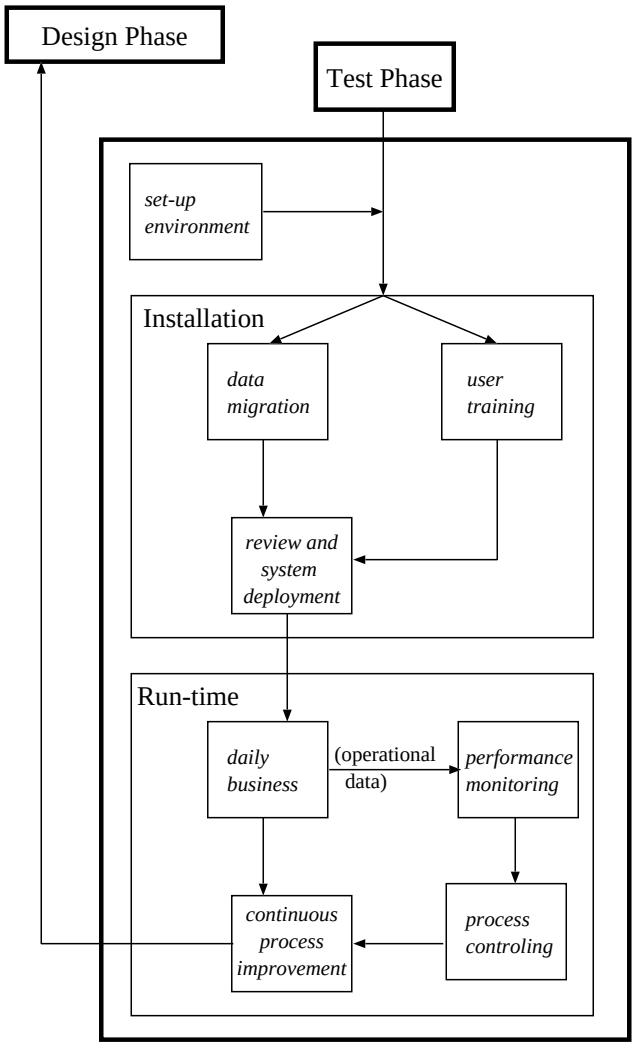

Figure 4.6: Operational Phase.

4.3 Case Studies Revisited

In Section 4.1, a set of problems in workflow projects were identified. Below we explain how the workflow application development methodology presented in the previous section helps in coping with these problems.

Problem 1 states that there are strong dependencies between the technical and organizational parts of the to-be business process models. For this reason, the methodology suggests that organizational and technical to-be modeling should be done collaboratively, and that the activities should exchange information constantly. This is done in the Design phase (cf. Figure 4.3), where the sub-phases during which the organizational model and the technical model are developed are performed in an integrated way, i.e., they are performed concurrently and they exchange information. In addition, multiple iterations of the modeling phases may lead to a concise and integrated model which represents both aspects properly.

System selection was often carried out in early project stages, resulting in Problem 2. In the workflow methodology presented, the system selection is based on the to-be business process model, which is created during the Design phase. Hence, the requirements of the workflow application with respect to workflow management system functionality are known when a system is selected. For example, the technical model created during the Design phase holds information on the integration of legacy applications, which can be used to select a workflow management system, capable of supporting the integration with little or no additional implementation effort. A requirement defined in the organizational model may be support of a certain organizational structure. This information can be used to select a workflow management system which is capable of supporting this structure. In general, the functional requirements are known when the system is being selected, which may lead to selecting more adequate systems.

The development of prototypes was identified as an important factor for the success of a workflow project, formulated as Problem 3. Instead of defining a dedicated prototyping phase in the workflow development methodology, prototyping is reflected by the evolutionary character of the overall process and by the extensive test phases.

An automatic transfer of business process models into workflow schemas proved to be unsuitable in most cases (cf. Problem 4). Although the workflow schema is based on the to-be business process model in the workflow application development methodology, an automatic translation of the corresponding modeling information cannot be recommended. Instead it is the task of the project team to use the to-be business process model and the information on the technical infrastructure of the organization to develop a workflow schema, which is expressed in the language of the workflow management system

selected.

Problem 5 stresses that the integration of legacy systems is a critical factor in workflow projects. For this reason, the Survey phase contains activities in which the characteristics of legacy applications are analyzed in order to identify possible restrictions concerning the integration aspect. To provide a high degree of stability and availability in legacy integration, the definition of corresponding test scenarios in the Test phase and extensive testing is recommended.

Many of the workflow management systems available are lacking performance and reliability if they have to cope with a large number of users and workflow instances, described as Problem 6. The workflow application development methodology helps in dealing with this issue by providing elaborate System Selection and Test phases. Based on the number of expected workflow participants and workflow instances, performance and reliability issues should always be key factors in the planning and conducting of Test phases.

4.4 Summary

In this chapter workflow application development processes are investigated. We believe that a thorough study of these processes can lead to more timely workflow projects and more adequate, usable, and reliable workflow applications. While the workflow application development methodology presented in this chapter is designed to avoid a number of problems which occurred in real world workflow projects, not all potential problems are solved by the methodology automatically: In workflow projects, there is no substitution for knowledgeable project managers, skilled developers, and efficient users. We remark that the methodology is a proposal, and real world usages may show its quality. However, since “Models are not right or wrong, they are more or less useful” [29], we believe that the workflow application development methodology presented is indeed useful in assisting project managers and participants in planning and conducting workflow application development projects.

Part II

Realization Concepts

This part discusses realization concepts for workflow management systems. In particular, concepts and methods to realize flexible and distributed workflow management systems are presented, based on the mathematical formalization of workflow schemas and workflow instances as proposed in Part I. After discussing a generic, workflow-based architecture, the conceptual design and the architecture of a first prototypical workflow management system are presented. The shortcomings of that prototype and additional recent research results motivate the conceptual design and prototypical implementation of a second, more elaborate prototype, which is in the center of Part II. Since a mathematical formalization is not sufficient as a basis for the conceptual design of a workflow management system, object-oriented design methods are used. Workflow schemas, workflow instances, and related artifacts are modeled as objects. Since the resulting class diagram specifies the structure of workflow-relevant objects, it amounts to a workflow meta schema. The workflow meta schema is designed to support dynamic adaptations of running workflow instances by changing the relationship of a workflow instance object to a workflow schema object. To control dynamic adaptations properly, correctness criteria using the mathematical formalization are introduced. The usage of the system is illustrated by a sample workflow application, which also introduces a workflow client application. The workflow client application can be configured to the functional requirements of different groups of persons involved in workflow applications, for instance workflow modelers, workflow administrators, and workflow participants.

Chapter 5

Project Overview

The research project started as an international cooperation in 1994, in which the University of Campinas, Brazil, and the University of Münster participated. Incidentally, it is an example of a research project whose topic changed dynamically at runtime. Setting out in the area of database support for scientific applications, it soon was realized, however, that workflow support may indeed be very beneficial for these application, so a "Workflow-based Architecture to support Scientific Applications", the generic WASA architecture, was developed. From a high-level point of view, three phases can be identified in the project:

- In the first phase (1994–1995), requirements of scientific applications in molecular biology and geo-processing were analyzed, and the notion of scientific workflows was introduced. A generic architecture for a workflow-based, integrated working environment for scientists in natural sciences domains was proposed. The results of this phase are reported in [5, 6, 71, 111].

- In the second phase (1996–1997), a first prototype was designed and developed. By dropping the build-time versus run-time approach and by using database and Internet technology, it provides some flexibility and platform independence [115, 109].

- The main goal of the third phase (1997–1999) is to develop an object-oriented workflow management system, whose main properties are dynamic adaptability of running workflow instances based on formal correctness criteria, re-use of workflow schemas and persistent and distributed workflow execution control without a centralized workflow engine [106].

5.1 Scientific Workflows and Generic Architecture

As indicated above, the project started in 1994 with the aim of supporting database applications in the natural sciences, mainly in molecular biology [71] and geo-processing [6]. By analyzing these application domains, we found that while the scientific communities have developed a rich set of application-specific data management and data analysis tools, there is little support for modeling and conducting application processes. However, it was observed that scientific work was typically carried out in complex working procedures, in which numerous persons, devices as well as application programs and domain-specific data are involved. Instead of relying on information system support, scientists in laboratories tend to maintain handwritten records of the activities planned and conducted, laboratory notebooks. These laboratory notebooks can be regarded as repositories for scientific working groups, which hold information on the working procedures as well as on application programs and scientific devices required to perform experiments.

In the context of laboratory information management systems, application processes have been identified, and requirements for suitable workflow support for these applications have been established in [87, 86]. In these settings, the repetition of process steps is a typical pattern, for instance to redo analysis steps which were conducted unsuccessfully. Another interesting aspect of application processes in laboratory environments is the dynamic creation of new process steps or even of complete new processes. This is due to the fact that during the analysis of a sample, certain contaminations may be detected, which is prompted by additional analyses conducted on the same or on related samples. The kinds of contamination and, consequently, the process steps to take, in general are not known in advance, rendering dynamic adaptations an important factor for workflow support in laboratory environments.

Given these situations, scientific work can benefit from information system support for documenting scientific working procedures, helping in planning, conducting, and documenting them. These observations and discussions with domain experts led to the idea to make use of workflow technology in the context of scientific applications. The term scientific workflow is used to indicate the abstraction of application processes in scientific domains. Thereby we do not mean to perform automatically scientific work, which of course is inherently creative in nature and, hence, cannot be performed automatically. On the contrary we aim at planning, describing, controlling, and monitoring the execution of application processes in scientific environments.

Workflows in scientific settings are flexible in nature: Typically a scientific experiment starts with some activities to be carried out. During the execution of these steps, new knowledge is generated which allows to decide on the future steps of this experiment.

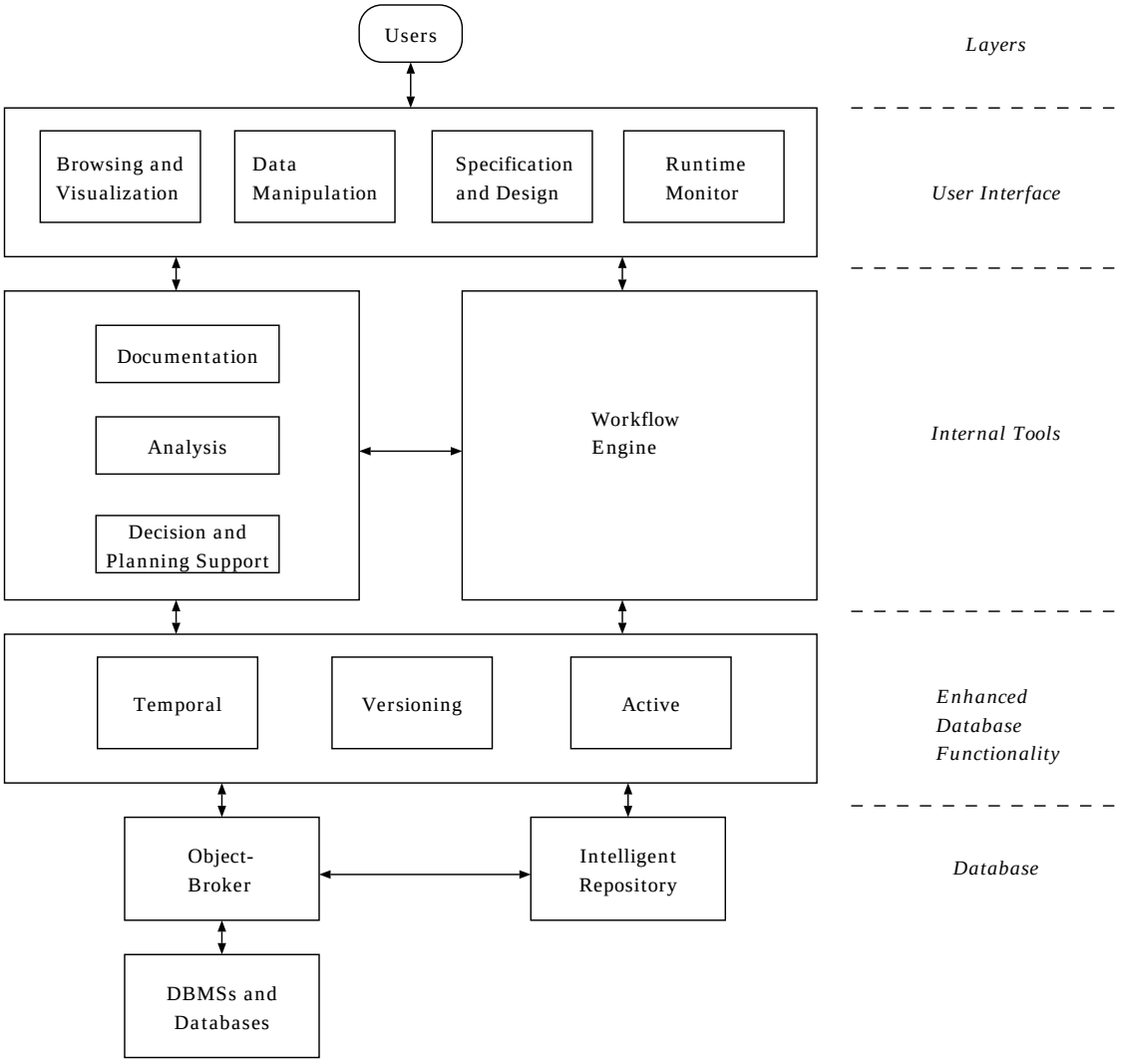

Figure 5.1: Generic Workflow-Based Architecture.

While this discussion idealizes the situation, dynamic change of running workflows is an important functionality for workflow support in scientific domains. Hence, the project from the beginning aims at supporting dynamic changes of running workflows, which are also described as dynamic adaptations of running workflows to a new workflow schema. It is interesting to notice that flexibility in general and dynamic adaptations in particular are no longer limited to scientific applications. As was indicated above, quickly evolving markets and technologies, the ability to react rapidly to changes in the environment of workflows nowadays is an important success factor for organizations to get and remain competitive.

Returning to the discussion of the first phase of the project, flexibility, platform indepen-

dence, and the integration of existing domain-specific data manipulation and data analysis tools were identified as important aspects of scientific workflows. A generic architecture — the WASA architecture [5] — was proposed to provide a workflow-based, integrated working environment for scientists aiming at supporting the required functionality. The generic architecture is shown in Figure 5.1; in this architecture four layers of functionality can be distinguished, each of which consists of a set of components.

The User Interface Layer is the top-level layer of the generic architecture. It consists of four main functional blocks which communicate with each other and with the internal tools. The Specification and Design facility provides users with tools to specify and design experiments as workflows. It provides access to previously designed workflow schemas for workflow schema re-use. The Data Manipulation facility provides users with means for accessing and manipulating domain-specific data. This includes navigation through reported experiments and their results as well as the invocation of domain-specific data analysis procedures. The Visualization facility allows users to browse and visualize different kinds of application data as well as workflow-relevant data. Finally, the Runtime Monitor can be used to control the execution of workflow instances and to monitor running workflow instances.

The Internal Tools Layer consists of the workflow engine and of a set of auxiliary managers to support experiment specification, documentation, and execution. The workflow engine is the core component of the generic architecture. It provides the functionality required for the controlled execution of workflows instances. The Documentation manager provides the means to document the conduction of experiments. It also allows recording of relevant events that occurred during the specification or execution of the experiment. The Analysis manager is responsible for managing the interface to application-specific analysis procedures, and for controlling their execution as requested by users. We imagine this to be a loose coupling only, in the sense that this manager will generally know where to find relevant procedures and analysis tools, to export input data to such tools, and to import their results and findings into the system.

The Enhanced Database Functionality Layer consists of a set of mechanisms that provide the relevant advanced functionality in a system-independent way. Three types of functionality are expected to be of particular importance here: Temporal support, e.g., for keeping track of past experiments, activeness, e.g., for controlling the execution of workflows, and version management for dealing with versions of data items as well as with versions of workflow schemas. Finally, the Database Layer will be handled through a CORBA Object Request Broker [76, 80]. This broker is responsible for retrieving the appropriate data and control descriptions from stored data sets, and is expected to support a variety of data models and query languages. However, the system creates application-

specific data and workflow-specific data. The former is stored in pre-existing databases, and accessed via the object broker; the latter goes into a data repository and is exclusively accessed by the system.

While the generic architecture was never implemented in a running system, it contains many aspects of an integrated environment for advanced workflow support. Many of the ideas which led to the development of the prototypes can be found, either directly or indirectly, in the generic architecture.

5.2 First Prototype

The second phase of the project was carried out during 1996 and the spring of 1997. During this phase a first running prototype was designed and implemented. Its main goals are re-use of workflow schemas, platform independence of both workflow clients and the workflow engine as well as a limited form of flexibility in executing workflow instances.

5.2.1 Design Decisions

The goals of workflow schema re-use, platform independence, and flexibility led to the decisions (i) to use an interpretation-based approach to workflow instance control, (ii) to store workflow schemas in a database and (iii) to use Java and web technology on both the workflow client and the workflow server sides.

Most workflow management systems are based on the traditional built-time versus run-time approach: During the built-time workflow schemas are specified completely. Workflow schemas are then used at run-time to instantiate and control the execution of workflow instances. Since dynamic modifications require more flexibility than this approach has to offer from a technological point of view, an interpretation-based approach is more adequate. In an interpretation-based approach, the workflow management system interprets workflow schemas successively during the execution of workflow instances. The step-by-step interpretation of workflow instances allows to react to changes in the workflow environment, which were not anticipated at workflow modeling time. The prototype is based on the interpretation approach. The execution of workflows is controlled by a workflow engine, which retrieves workflow schemas during the run-time of the workflow instance. In this approach, workflow schemas can be changed, and running workflow instances can use modified workflow schemas. Approaches to workflow execution control are discussed in more detail in Section 7.2.

In order to store workflow specifications persistently and to enable the re-use of workflow schemas as sub-workflow schemas in multiple complex workflow schemas, workflow schemas are stored in a database. During workflow modeling, existing workflow schemas can be assembled to form new complex workflow schemas. Re-use is important in this context since it reduces the workflow modeling overhead. In addition, changes applied to a given workflow schema are automatically propagated to all occurrences of the workflow schema in multiple complex workflow schemas. In the prototype described in this section, workflow schemas are stored in a relational database that maintains relations for the components of workflows. In particular, there are relations for workflow schemas, input and output parameters, data and control connectors, variables, roles, agents, and other artifacts. In the database representation, creating a workflow schema amounts to inserting tuples into database relations, while modifying a workflow schema is done by appropriate updates against the database. Sub-workflow schemas are retrieved successively while the workflow executes, which allows to perform changes to the parts of the workflow, which have not yet been instantiated. The conceptual database design, the system architecture, and the operation of the workflow engine are discussed in more detail below.

5.2.2 Conceptual Model and Database Schema

In order to provide reliable storage of and efficient access to workflow schema information, workflow schemas are stored in a database. We opted for a relational database system for reasons of performance, stability, and local availability. To develop a database representation of workflow specifications and related artifacts, conceptual database design techniques are used [107]. In conceptual database design, a conceptual model is developed, typically using a graphical formalism. Entity-Relationship diagrams are widely used for this purpose [16].

Based on an Entity-Relationship diagram, a logical relational database schema can be created using a set of transformation rules [81], and the relational database schema can be implemented, using the language constructs provided by the database management system. Issues in physical database design are addressed next, for instance the number and size of table-spaces and the mapping from logical relational tables to physical storage. This thesis concentrates on the conceptual design of representing workflow schemas and the design of the database schema; issues of physical database design are not taken into consideration.

To create an Entity-Relationship diagram modeling the entities and relationships relevant for the design of workflow schemas, the following considerations are appropriate:

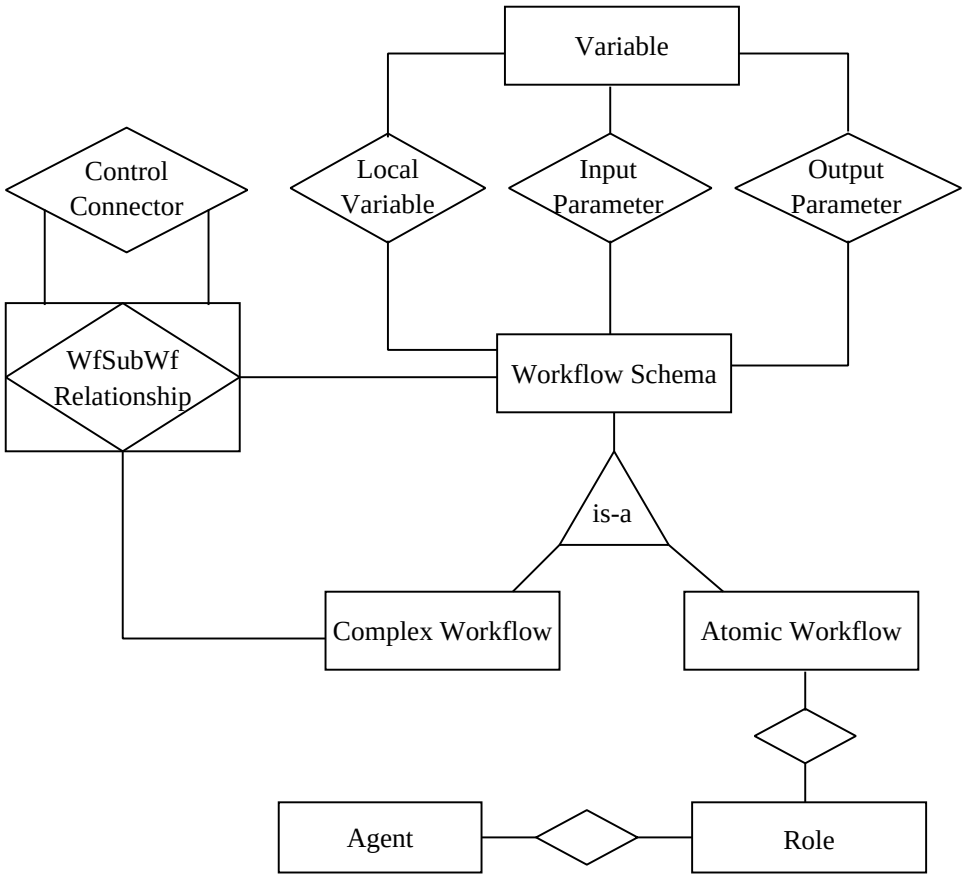

Figure 5.2: Simplified Conceptual Model, specified by Entity-Relationship Diagram.

Workflows are specified by workflow schemas. Each workflow schema may occur in multiple other workflow schemas as sub-workflow schema. In order to support the information aspect properly, variables and parameters are used. In particular, workflows can be assigned local variables, input parameters and output parameters.

WorkflowSchema	=	{WorkflowId, Description }
ComplexWorkflow	=	{WorkflowId}
WfSubWfRelationship	=	{WorkflowId, SubWorkflowId }
Variable	=	{WorkflowId, VariableId, VariableSchema, IsInputParameter, IsOutputParameter}
ControlConnector	=	{SuperWorkflowId, SourceWorkflowId, TargetWorkflowId, Condition }
AtomicWorkflow	=	{WorkflowId, RoleId, ApplicationId, Automatic }
Role	=	{RoleId}
Agent	=	{AgentId, Password}
RoleAgentAssignment	=	{RoleId, AgentId}

Figure 5.3: Database Schema to Represent Workflow Schemas.

The hierarchical structure of complex workflows is represented by a workflow sub-workflow relationship between complex workflows and workflows. Since occurrences of a given workflow schema in the context of different complex workflow schemas can have different control flow and data flow constraints, these constraints are related to workflow sub-workflow relationships as opposed to workflow schemas directly. In the organizational context, a simple role concept is supported. Notice that the workflow sub-workflow relationship is considered an entity, allowing control flow connectors to be defined conveniently by pairs of these relationships. However, this construct can be regarded a shorthand notation, which can be expanded by introducing an entity type for the workflow sub-workflow relationship and additional relationships, linking that entity type to workflow schema and complex workflow, respectively. These considerations will re-occur and are explained in more detail when presenting the object-oriented design of a workflow meta schema in the next chapter. For each atomic workflow a role is defined, which in turn is resolved to particular agents when the workflow executes. Based on these considerations as far as the conceptual design is concerned, a simplified version of the Entity-Relationship diagram is given in Figure 5.2.

By using well-known techniques from conceptual database design, the conceptual

model can be transferred to a logical database schema. In particular, entities and relationships are mapped to relational tables and relationships between tables. The database schema for the prototype is given in Figure 5.3. For each relation, the name of the relation and the set of attributes are provided. Attributes which belong to the key for that relation are underlined. Given that database design, workflow schemas can be added to the workflow schema database by inserting tuples in the respective relations. During the run time, workflow schema information is retrieved from the database by issuing select commands against the workflow schema database. More detailed information is provided in [115].

5.2.3 System Architecture

The design decisions taken lead to the system architecture shown in Figure 5.4. The prototype is based on a client/server architecture. The workflow engine is a multi-threaded Java program which reads workflow schemas from a relational database, instantiates workflow instances, and controls their execution. Internally, the Java program is composed of a workflow engine (called Kernel in Figure 5.4) as the core component and the database server which is used to access application data stored in the underlying database. Both components are connected to the database by a Java Database Connectivity (JDBC) interface.

A workflow modeling tool allows the graphical specification of workflow schemas. Workflow schemas may be built from scratch or existing workflow schemas may be reused, e.g., as sub-workflows in different complex workflows. On start-up, the workflow modeling tool connects to the database via a JDBC interface. To store workflow schemas, the workflow modeling tool generates SQL statements which insert tuples representing the workflow schemas into the workflow schema database. A detailed description of the workflow modeling tool and of a workflow monitoring tool, which allows to monitor the progress of running workflow instances, can be found in [28].

When a workflow is instantiated, the workflow engine retrieves the respective workflow schema from the workflow database. The workflow engine then creates a thread which is responsible for controlling the execution of that particular workflow instance, and the workflow instance starts. When a complex workflow is launched, only that sub-workflow is retrieved that has to be executed first; subsequent ones are only retrieved from the database when needed. This approach has two advantages: First, it minimizes workflow start-up time and loading overhead, since only the workflows are instantiated which are actually required during a particular complex workflow. Second, it allows to change the specification of sub-workflows while the complex workflow runs, as explained above.

Users access the workflow system using workflow clients. The basic functionality of

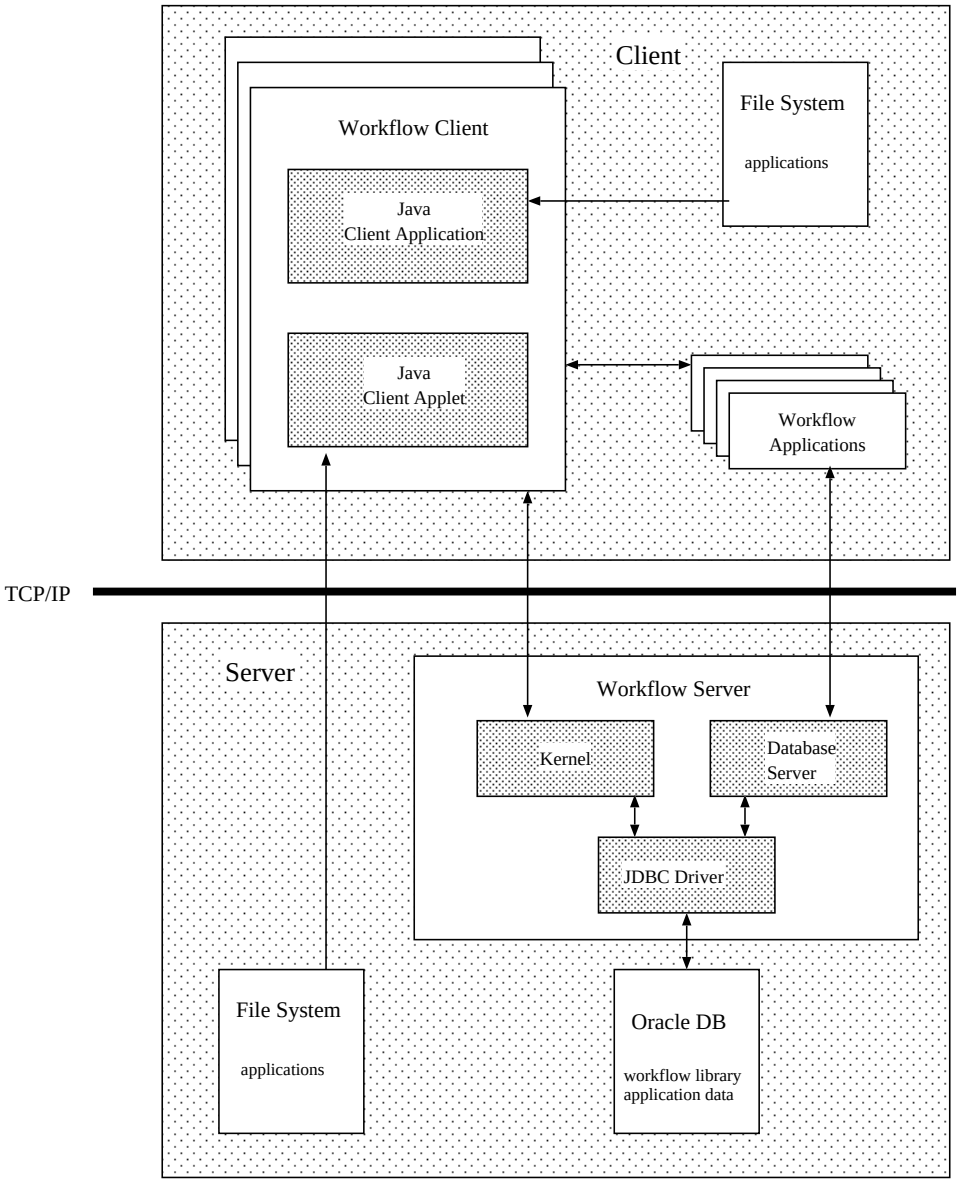

Figure 5.4: System Architecture.

a workflow client is to inform workflow participants of activities to perform. There are two implementations of workflow clients: Java applications and Java applets. In order to run workflow clients as Java applications, a Java runtime system has to be available on the client side. Since there are no restrictions on the accesses of Java applications, workflow clients may access local data and may start external applications. Using Java applets, the availability of a standard web browser suffices on the client side. The Java applet is downloaded to the client web browser and interpreted by the browser. Both implementations communicate with the workflow server using the TCP/IP protocol. However, security restrictions apply to Java applets, e.g., access to local data and starting external applications may not be performed by workflow clients implemented by Java applets. Besides the platform independence on the client side, we also installed the workflow server on SUN workstations and on Windows NT workstations, using different commercial relational database systems.

5.2.4 Critique

While the first prototype fulfills some of the goals for flexible workflow management, there are a number of deficiencies and shortcomings. These as well as new concepts serve as motivations for the design and implementation of the second prototype. The positive aspects of the first prototype are summarized as follows:

- *Re-Use of workflow schemas:* Workflow schemas can be re-used as sub-workflow schemas in an arbitrary number of complex workflows. This property allows to define a workflow schema once to use it multiple times. When there are changes to a workflow schema, all occurrences of that schema are changed automatically, since they reference the same workflow schema.

- *Platform independence:* By using Java and Internet technology, the prototype provides a high degree of platform independence, since it is feasible to access the system via web browsers from a variety of hardware and software platforms connected to the Internet. In addition to clients, the server is also written in Java. Hence, the server is also platform independent, provided a relational database to store workflow schemas and a JDBC driver to access the database from the Java workflow server are available. We have successfully used SUN and Windows platforms, running Oracle 7 and MS Access database systems, respectively. Hence, not only the workflow clients provide platform independence, but also the workflow server.

- *Dynamic Modifications:* The prototype supports a limited form of dynamic modifications of running workflow instances. Since the sub-workflows of a complex

workflow instance are retrieved from the workflow database and created during the execution of complex workflow instances, changes to sub-workflow schemas are used in complex workflow instances, provided the change occurs before the sub-workflow schema is retrieved from the database.

The prototype was a first approach to develop a flexible workflow management system. Therefore it is not surprising that it exposes a number of deficiencies and shortcomings, which will serve as motivation for the design of a second prototype; the shortcomings of the first prototype are described as follows:

- *Limitations in Dynamic Adaptation:* While the prototype allows changes to sub-workflow schemas while they are not instantiated, structural changes of complex workflows are not supported. This is a severe restriction of dynamic adaptations, since changes of the process structure is an important feature of flexible workflow management. In addition, correctness checks are restricted to syntactic correctness. This means that a sub-workflow schema i of a complex workflow can be changed to sub-workflow schema j if their input and output parameters match. No additional consistency checks are provided.

- *Little Fault Tolerance:* In general, fault tolerance is an important issue in workflow management systems design, because mission critical business processes are typically supported by workflow technology. If a workflow application fails, then the system must be able to restore a consistent state of all running workflow instances. In the prototype, workflow schemas are stored persistently in a relational database. Workflow instances, however, are not. Workflow instances live as threads inside the workflow server. If the workflow server fails or there is a system error with loss of main memory content then the state of running workflow instance cannot be restored properly.

- *Centralized Workflow Engine:* The prototype uses a centralized workflow engine to control the execution of workflow instances. As already mentioned, each workflow instance is represented by a thread in the workflow server program. The threads run concurrently, so that a given workflow instance can proceed independently of other workflow instances. In large-scale workflow applications, this means that there are hundreds of thousands of treads running inside the workflow server. This centralized approach to workflow instance execution control may result in performance issues. In addition, the centralized workflow server is a single point of failure. This means that the operation of the complete workflow application and all its workflow instances depends on the operation of the centralized site, i.e., the workflow server.

5.3 Lessons Learned

As discussed in the previous section, the first approach provides a rather limited form of dynamic adaptations. Since flexibility and dynamic adaptation has been an important research topic in the project from the beginning, the novel system should provide much more functionality in this context. First of all, structural changes to complex workflow schemas should be possible, i.e., adding or deleting sub-workflows to a given complex workflow or adding or deleting execution or data constraints should be supported. Dynamic adaptations should be controlled, i.e., consistency criteria should be defined which can be used to determine whether a given workflow instance can be adapted to a particular workflow schema.

Workflow execution control was centralized in the first prototype. Since workflow instances are inherently executed in distributed settings, a distributed approach to workflow execution control is more appropriate than a centralized one. The most obvious reason for this fact is scalability and fault tolerance. In large-scale workflow applications, a centralized workflow server is likely to become a major performance bottleneck. With respect to fault tolerance, in a centralized setting the operation of the complete workflow application depends on the functionality of a single component, i.e., the centralized workflow engine. This is not a desirable functionality in distributed applications. In distributed workflow management, only certain parts of the workflow application are affected when a site goes down. In particular, workflows which execute in other sites can continue execution, ideally without any disturbances. We remark that this feature applies not only to different application processes, represented by different top-level workflow instances, but also to workflow instances of a single top-level workflow.

Maybe the most important reason for distributed workflow control is local autonomy. Hereby we mean that the organizational entity which is responsible for the execution of a complex workflow can control the execution locally under their own supervision. For instance, a department is responsible for the execution of a given complex workflow, which by itself is a sub-workflow of a complex workflow on a higher level. For instance, in an order-processing workflow, the warehouse department may be responsible for performing a complex workflow to check the inventory of the warehouse. Distributed workflow management allows to execute the workflow “check-inventory” under the supervision of the warehouse department. Thus, the organizational assignment of a complex workflow to a specific department is reflected by migrating the complex workflow instance to the department workflow server. To summarize, distributed workflow execution control (i) makes possible large-scale workflow applications, (ii) maps organizational competence structures to the system design, and (iii) enhances fault-tolerance.

Workflows in the first prototype are not persistent. This means that the state of workflow instances is maintained only in volatile memory. As a result, a system failure of the workflow server leaves workflow instances in an inconsistent state. Of course, non-persistent workflow management systems cannot be used in commercial settings. The novel system should provide persistence for both workflow schemas and workflow instances. Storing the state of workflow instances in persistent storage allows to restore the current state of workflow instances in a consistent way after a system crash has occurred. If terminated workflow instances are also stored persistently, then application-specific analysis procedures can be applied, for example, to gather information on the performance of particular workflow instances. This information can be used to improve workflow schemas during future process improvement phases.

As discussed in Chapter 4, the integration of external applications is an important aspect in developing robust commercial workflow applications. One important recent development in software engineering is component-based development based on business objects and open interfaces. The novel system should provide the ability to easily integrate business objects into workflow applications.

5.4 Summary

The WASA project has gone through three phases. In the first phase requirements of scientific applications in molecular biology and geo-processing were analyzed, and the notion of scientific workflows was introduced. A generic architecture for a workflow-based, integrated working environment was proposed. To prove the concepts, in the second phase a first prototype was developed. By dropping the build-time versus run-time approach and by using database and Internet technology, it provides some degree of flexibility and platform independence for both workflow clients and the workflow server. The design and implementation of the first prototype is sketched. In particular, the conceptual design of the database representation of workflow schemas and related artifacts is discussed. The system architecture shows the components on the client and server sides, respectively. The lessons learned during these early developments found their way into the conceptual design and implementation of the next prototype, which stands in the center of the third phase, discussed next.

Chapter 6

Conceptual Design of a Workflow Management System

This chapter introduces the conceptual design of a novel workflow management system; the chapter is organized as follows. First, the design goals are presented, and modeling alternatives for object-oriented workflow management systems are discussed. Section 6.3 introduces the conceptual design of the novel workflow management system by presenting the workflow meta schema and a state transition diagram, which is used to specify the behavior of workflow instances. The chapter completes with a discussion of the compliance of the formal foundation of workflow schemas and workflow instances and the object-oriented design presented.

6.1 Design Goals

In order to be suitable for a broad spectrum of workflow applications, there are different requirements that have to be met by a workflow management system [91]. While there are numerous requirements of this kind, we put our emphasis on the topics discussed below.

6.1.1 Dynamic Adaptation

Traditionally, workflow management deals with controlling the execution of application processes according to pre-defined workflow schemas [32, 67, 105]. This approach is well suited to support application processes with static control structures, i.e., application processes which are modeled once and which are executed in a routine fashion. In workflow projects in commercial and scientific settings, however, it turned out that while some application processes do have a static structure, others don't. From an application-

oriented point of view the need to react to new situations, which were not anticipated during workflow modeling time, led to flexibility requirements.

Since traditionally dynamic adaptability was not a topic in workflow management, the respective functionality is not adequately supported in commercial workflow management systems. This limitation is regarded today as one of the main obstacles for the deployment of workflow applications in real-world organization, because workflow participants and workflow administrators often encounter situations which have not been foreseen at workflow modeling time. To discuss the shortcomings of missing flexibility support in workflow management systems, a workflow participant is considered who encounters a situation, which was not anticipated during workflow modeling time. Consequently, the application process cannot be continued as was specified in the workflow schema. To cope with the new requirements which emerged while the application process runs, the workflow participant performs certain activities outside the scope of the workflow management system, for instance by accessing an information system to get hold of additional information. If modeled workflow activities are not required and the workflow management system does not allow to skip workflow activities then the workflow instance has to be stopped altogether, and the application process continues without workflow support. In any case, the workflow management is not informed about the conduction of the application process. As a result, a major aim of workflow technology, i.e., the controlled execution of application processes, cannot be realized. Obviously, this situation is not desirable since the workflow management system does not have any information on how the application process was actually executed.

We remark that long-running workflow instances fortify the need for dynamic adaptations. In the insurance domain, e.g., there are workflows which run for weeks, months or even for years. If and when new regulations are imposed in the insurance market or existing limitations are dropped, dynamic changes of long-running insurance workflows are inevitable. However, also workflows with a shorter duration can benefit from dynamic adaptations, for instance in the telecommunication domain. Due to dropped regulations in that market in Europe, new consortia are created, and new services and new pricing schemes are created constantly. These services and pricing schemes to a large extent are implemented in business processes of telecommunication companies, a considerable part of which may be supported by workflow technology. In order to implement these changes quickly, the organization will not only apply new process specifications to new workflow instances but also to running workflow instances. In fact, to react quickly to changes in the market environment is an important success factor for today's commercial organizations or, as an analyst put it, "today not the big eat the small, but the fast eat the slow".

While the exact definition of flexibility in workflow management systems is still under

discussion [101, 61], it is widely accepted that dynamic adaptations of running workflows is an important feature of a flexible workflow management system [84, 27, 98]. In the remainder of this thesis based on the formal foundation, a conceptual design and a prototypical implementation of a workflow management system are proposed which supports controlled dynamic adaptations of running workflow instances to new workflow schemas.

6.1.2 Distribution and Scalability

Traditionally, workflow management systems are client/server systems such that the server corresponds to a workflow engine, and workflow participants access the system using workflow clients as graphical user interfaces. Considering workflow executions, this approach can be conceived as distributed since the work performed during the workflow is done in the client side, and in general there are multiple clients involved in workflow executions. Typically to perform a workflow activity, a workflow participant starts an application program, which is executed on the client machine or on another machine in the network. However, as far as controlling the execution of workflows is concerned, the client/server-approach is in fact a centralized one: The workflow engine is a centralized server, which controls the execution of all workflows in a given organization in a single, centralized site. It does so by communicating with workflow clients, which represent participants and/or application programs invoked during workflow executions. In large scale workflow applications, the centralized site is likely to receive a heavy load, which may result in scalability problems. Since workflow management is inherently performed in distributed settings, it it not surprising that a centralized solution to workflow execution control generates a set of issues: Besides the fact that the centralized workflow engine can become a performance bottleneck of the system, it is a single point of failure, such that the availability of the workflow application depends on a single site. This is not a desirable feature in a distributed environment. Regarding performance and fault tolerance issues, for these kind of application it would be much more adequate to use a distributed approach instead of a central one.

6.1.3 Persistence

The success of organizations nowadays relies to a considerable extent on the availability of its information infrastructure. In the database context, this has led to the development of elaborate concurrency control and recovery techniques, aiming at providing execution guarantees in the presence of multiple concurrent users and system failures [7, 37]. Providing this kind of functionality is also important for workflow management systems. In

particular, a system failure should not leave running workflow instances and manipulated data in an undefined state. In contrast, up-to-date information on the state of running workflow instances has to be kept in stable storage in order to be able to continue interrupted workflows after the system is restarted. A key prerequisite to develop a fault-tolerant workflow management system which implements a functional workflow restart procedure is to maintain explicit state information of workflow instances and accompanying data on running workflow executions in stable storage.

6.1.4 Re-use of Workflow Schemas

To allow efficient workflow modeling and to minimize redundancy in workflow modeling, it is important that workflow schemas are defined once to be used multiple times. In particular, each workflow schema can be used as a sub-workflow schema in multiple complex workflow schemas. For instance, a workflow schema “transfer funds” in a banking environment can be a sub-workflow schema in complex workflow schemas “pay mortgage” and “buy stocks”. Hence, reusing workflow schemas minimizes redundancy in workflow modeling. In addition, when workflow schemas have to be updated (for instance due to new company regulations or due to the installation of new external application programs) just one workflow schema has to be updated to implement the change in all occurrences of that workflow schema. It is advisable that organizations maintain workflow schemas in a workflow schema library, which serves as an important repository for the knowledge of the organization as far as application processes and their formalization as workflows are concerned. By accessing the workflow schema library, the efficiency of workflow modeling can be enhanced since workflow modelers can assemble complex workflow schemas from workflow schemas stored in that library, as opposed to starting from scratch each time a new workflow schema is created.

6.1.5 Integration

An important functionality of a workflow management system is the integration of existing external application programs (often legacy software systems) in workflow applications. As it turns out, the integration of external application programs and data sources in workflow applications proves to be a crucial success factor for a fair percentage of workflow projects, as was discussed in Chapter 4. Legacy systems often do not provide standardized interfaces to access data and functionality, so that proprietary solutions have to be developed to access legacy systems in workflow applications. One approach to remedy this situation is the recent interest in component-oriented software development based

on independent software components, which can be assembled to complex software systems. In the context of distributed object systems, probably the most important approach is the quest of the OMG to define Business Objects on top of the object-oriented middleware standard CORBA. Business objects perform application specific tasks, and data and functionality of business objects can be accessed via standardized interfaces, specified in CORBA Interface Definition Language. A workflow management system should support these new standards from the beginning to cater for a streamlined integration of business objects in workflow applications.

6.2 Modeling Alternatives

In this section, different approaches to an object-oriented modeling of workflow management systems are discussed. An object-oriented approach is suitable in the workflow context, since workflow management systems are complex software systems which benefit from the abstraction capabilities provided by object-orientation. Furthermore, workflow management systems are typically deployed in distributed and heterogeneous settings. By providing a consistent object model of a workflow management system, the system can be developed using object-oriented middleware, which is capable of handling distribution and heterogeneity issues.

Given the complexity of the domain, it is not surprising that there are numerous alternative object models for workflow schemas and workflow instances. In this section, some of these alternatives are discussed and evaluated with respect to the design goals presented. For each of the alternatives shown, a rather simple object model is discussed, containing workflow schemas and workflow instances. Since this section aims at discussing the basic properties of different modeling alternatives, other workflow components, for instance, control flow constraints, data flow constraints and information on the organizational and technical environment of the workflow execution, are not modeled explicitly.

6.2.1 Workflow Schemas as Individual Classes

In the first approach, workflow schemas are modeled as individual classes. For instance, the workflow schema to specify a retail workflow is defined in a class process-order. This class contains information on the structure of the workflow, on its sub-workflows, and their execution constraints. Workflow instances are objects of this class, i.e., the workflow schema is represented by the class definition, and workflow instances are objects in the respective workflow class. For example, workflow instances *process-order(Miller, Socks)*,

process-order(Smith, Ties), and *process-order(Taylor, Shirts)* are objects in the process-order class. In this modeling alternative, the hierarchical structuring of workflows in complex workflows and sub-workflows can be modeled by class aggregation, i.e., the complex workflow class has an aggregation relationship with all its sub-workflow classes. The execution constraints of the sub-workflows are stored in the complex workflow class.

As pointed out in Section 6.1, an important aspect of workflow modeling is the re-use of workflow schemas. In this modeling alternative, workflow schema re-use can be modeled by multiple aggregation relationships between classes. In organizations with many workflow schemas and with complex structured workflows, the class diagram tends to be very complex, hard to understand, and hard to maintain. In this alternative, the structure of a workflow is specified by the structure of the workflow classes involved. In this modeling alternative, aggregation relationships connect workflow classes if and only if there is a nesting relationship between workflows of these classes. Considering a dynamic change, in which a sub-workflow is added to a complex workflow, a new aggregation relationship has to be introduced between the workflow classes involved. The dynamic aspect forces this relationship to be added while workflows in this class are running. Adding structural information to classes, which do have active instances imposes considerable difficulties from a systems management point of view, since schema changes cannot be handled adequately and with ease in today's programming environments.

If the system can handle structural changes of a workflow class, all workflow instances of that class are affected by that change. This behavior is not desirable in general, since the dynamic change could have been meant to affect just a single workflow instance or a few workflow instances. Dealing with this problem requires a new workflow class to be created for the workflow schema generated by the dynamic adaptation. In addition, the workflow instances which shall be changed to reflect the dynamic adaptation have to be transferred to this class, involving considerable overhead due to object migration.

In summary, using individual classes for workflow schemas has a number of disadvantages with respect to the design goals presented above:

- Re-use of workflow schemas incurs considerable overhead due to complex and hard to maintain workflow class structure in large scale workflow applications.

- Dynamic adaptations are reflected by changes to the class structure of workflow specification classes. Schema changes in general are hard to manage. The problem is aggravated when a subset of the workflow instances have to be changed dynamically, resulting in complex object migration issues.

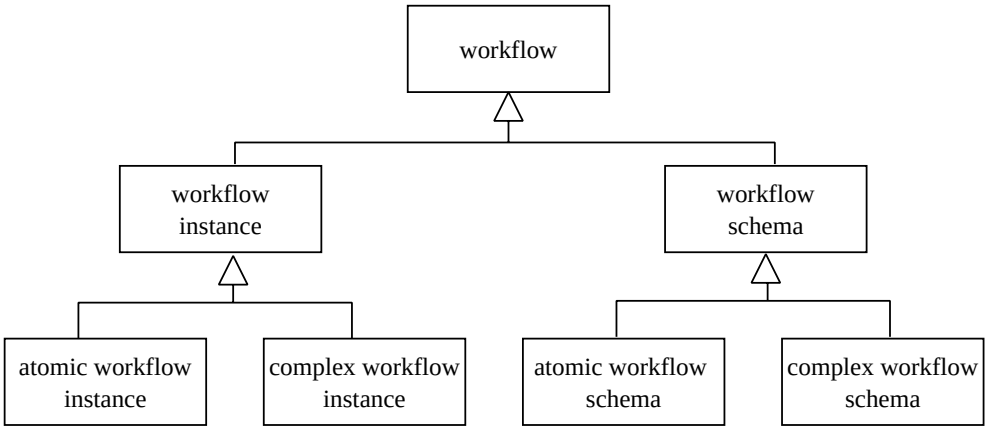

Figure 6.1: Generic Workflow Classes using Single Inheritance.

6.2.2 Generic Workflow Schemas

The second modeling approach is based on generic workflow classes. In the generic approach, workflow schemas are objects, stored in a single workflow schema class. In terms of levels of abstraction, the workflow schema class represents a workflow meta schema, since it specifies the structure of workflow schemas. The main difference to the first modeling alternative is the fact that workflow schema information is stored in data objects, not in classes.

This approach has a number of advantages. First of all, it supports workflow schema re-use well, since it allows to maintain the nesting relationship of complex workflow schemas with their sub-workflow schemas on the object level. In particular, workflow schema objects have relationships with other workflow schema objects, which specify the nested structure of complex workflow schemas. Hence, a given workflow schema can be re-used in many complex workflow schemas simply by referencing it in different complex workflow schemas. If there is a change to a workflow schema then all complex workflow schemas referencing the changed workflow schema will receive timely information about that change.

Workflow instances are maintained in a single workflow instance class. Since each workflow instance is controlled based on a workflow schema, the relationship between a workflow instance and a workflow schema is represented by a link between objects of these classes. In terms of flexibility issues and dynamic changes, this modeling alternative is more adequate, since the relationship between a workflow instance and a workflow schema can be changed while the workflow instance runs, namely by changing the respective object reference.

Based on generic workflow classes, there are two alternatives to model the hierarchi-

cal structure of workflows. These alternative approaches are based on single inheritance and multiple inheritance, respectively. While complex workflows and atomic workflows have common properties, they also expose considerable differences in their structure and behavior. Using object modeling techniques, their common properties can be maintained in a workflow class, with sub-classes representing atomic workflows and complex workflows, respectively. Hence, atomic workflow and complex workflow inherit properties of the workflow class and specialize it according to their requirements.

Figure 6.1 shows an object diagram using single inheritance, while the diagram shown in Figure 6.2 uses multiple inheritance. In the single inheritance diagram, a workflow class holds the common structure and functionality of workflow instances and workflow schemas. Since there are atomic and complex workflow schemas and workflow instances, the workflow instance class and the workflow schema class, both have sub-classes for complex and atomic workflows. This imposes redundancy since complex workflow instances and complex workflow schemas do have common structure and functionality, which is redundantly defined in two classes. Hence, the single inheritance alternative imposes redundancy in modeling workflow schemas and workflow instances.

When multiple inheritance is used, the redundancy issue disappears (cf. Figure 6.2): The complex workflow schema class inherits from the complex workflow class and from the workflow schema class. As can be seen in that figure, however, the resulting class diagram is rather complex. In a complete workflow meta schema, additional classes have to be taken into account, for instance classes for the complex structure of workflows and control flow, data flow, and organizational information. And for each of the components mentioned, there is a schema class and an instance class, representing the appropriate schema information and instance information, respectively. For instance, a control flow connector on the schema level represents a control flow between two sub-workflow schemas, while on the instance level, an object in the control flow instance class represents a control flow between sub-workflow instances. Given this situation, it should be obvious that the resulting object model in the multiple inheritance alternative has a highly complex class diagram, which may lead to significant maintenance overhead.

6.2.3 A Novel Approach

The generic alternative to modeling workflow schemas and workflow instances provides re-use of workflow schemas and a high degree of flexibility with respect to dynamic changes; workflow schemas are maintained as objects of a workflow schema class, and changes to objects are easier to handle than changes to classes or to class structures. Therefore the workflow meta schema follows the generic approach. However, the two

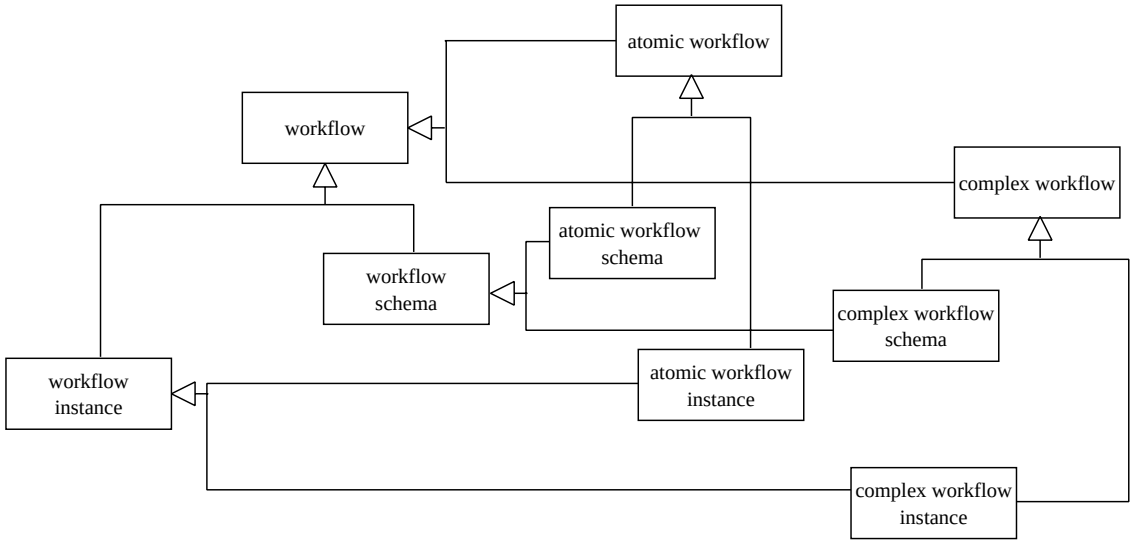

Figure 6.2: Generic Workflow Classes using Multiple Inheritance.

generic modeling alternatives presented above expose deficits, given the redundancy of the single inheritance object diagram (Figure 6.1) and the complexity of the multiple inheritance object diagram (Figure 6.2).

During the evolution of the workflow meta schema numerous modeling approaches have been discussed, the single inheritance model being one of them. We extended the object diagram shown in Figure 6.1 with other workflow relevant classes, for instance hierarchical relationships, control flow, parameters, data flow, agents, and roles. As indicated earlier, not only workflows but also these accompanying entities have a schema and an instance aspect. For instance, workflow schemas and workflow instances have hierarchical relationships; control flow and data flow appears in both workflow schemas and workflow instances; the same holds for parameters, data flow, agent information, etc. Hence, the object diagram tends to consist of two parts, one of which models the schema part, the other of which models the instance part. As a consequence, the resulting object model is mirror-like, with one part describing schema information, the other describing instance information.

There is a simple yet adequate solution to these issues: Workflow schemas and workflow instances are represented by a single class, the workflow class. Hence, workflow schemas and workflow instances are objects of a single workflow class. This decision solves the difficulties of the other modeling alternatives. In particular, issues of redundancy and complexity are solved since there is one class structure which includes both schema and instance information. We adopted this approach for workflow objects and

for accompanying objects like control flow objects, parameter objects, data flow objects, etc. In our approach, however, a new issue is introduced: The system has to know which object represents a workflow schema and which object represents a workflow instance. This issue is handled by introducing states “schema” and “instance” of workflow objects. Based on this general approach, the next section goes into the details of the object-oriented modeling.

6.3 Workflow Meta Schema

This section presents the conceptual design of a workflow management system. Based on a formal mathematical characterization of workflow schemas and workflow instances as proposed in Section 3.2, this section uses object modeling techniques to develop an object model, which is more appropriate for system development than a mathematical characterization is. In the object-oriented approach, relevant objects of the application domain are represented by objects in the software system, encapsulating their structure and behavior. Since the domain of interest is workflow management, workflow schema objects, workflow instance objects, and objects describing the organizational and technical execution environment of workflows are in the center of attention.

The workflow meta schema is shown in Figure 6.3. It is presented using the Uniform Modeling Language (UML) [79], an object-oriented modeling and design language. In UML class diagrams, classes are represented by marked rectangles; other important language constructs, such as generalization and aggregation, relationships between classes, and cardinalities are introduced in Figure 6.3. The workflow class is the central class in the workflow meta schema; it contains workflow objects which are either workflow schema objects or workflow instance objects. Workflows can be either atomic or complex, and atomic workflows can be executed either automatically or manually. This property of workflows is reflected in the workflow meta schema by defining complex workflow and atomic workflow as sub-classes of the workflow class, and automatic and manual as sub-classes of the atomic workflow class. The workflow hierarchy (i.e., the relationship between a complex workflow and its sub-workflows) is modeled by the WF-SubWF Relationship class, which defines a relationship between a complex workflow and a workflow, which can be complex or atomic.

As discussed above, workflow schemas and workflow instances are identified by different states of workflow objects. We remark that no workflow object changes its state from “schema” to “instance” or vice versa. There are constraints on relationships of workflow instances and workflow schemas, which can be specified in UML. One constraint

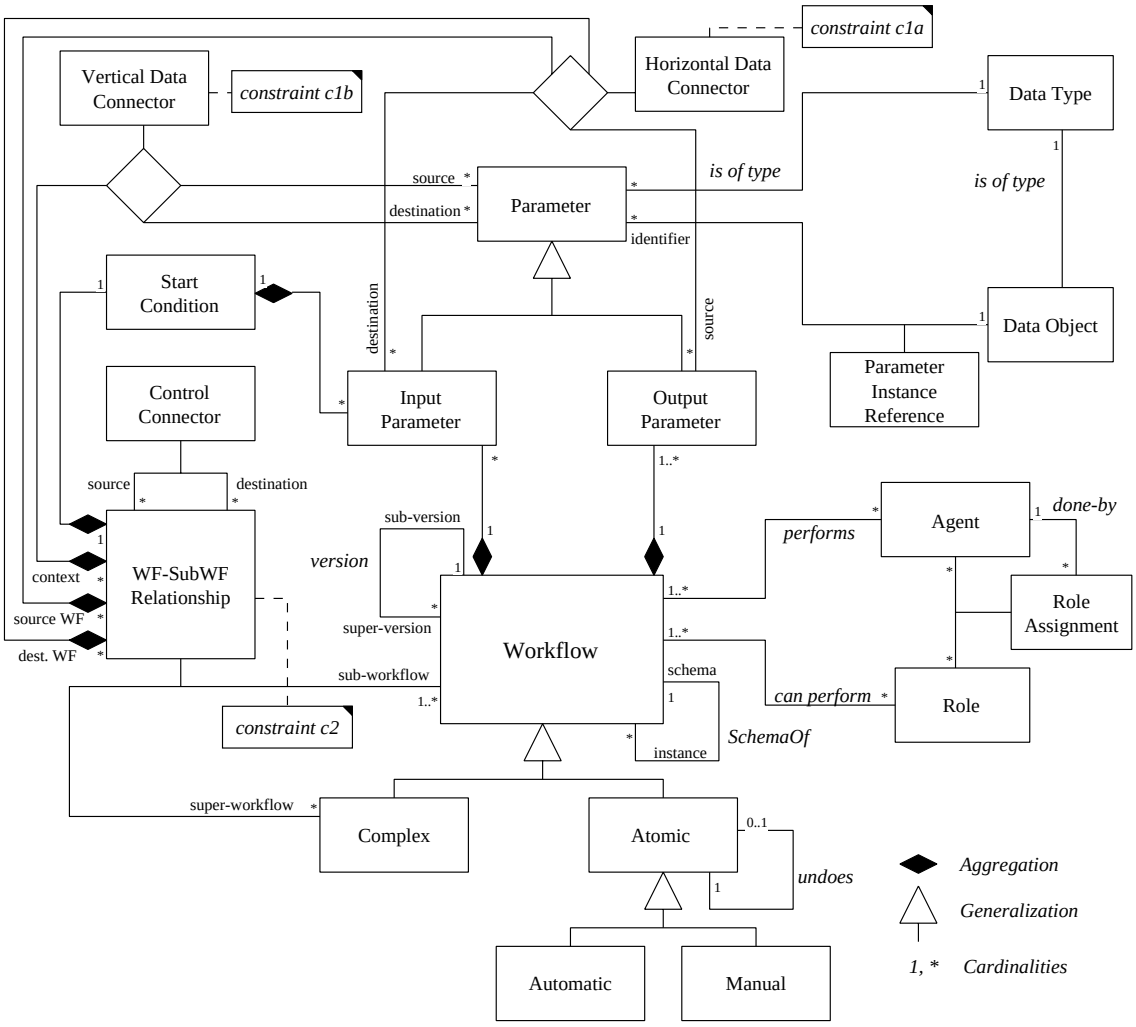

Figure 6.3: Workflow Meta Schema.

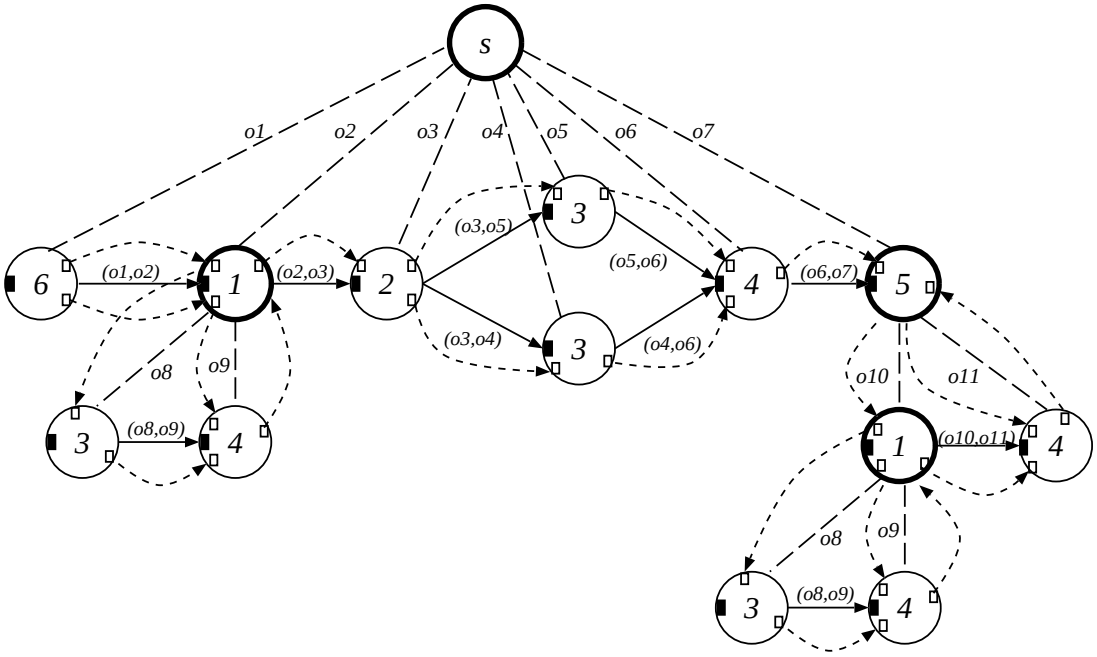

Figure 6.4: Representation of Complex Workflow Schema in Workflow Meta Schema.

involves the structure of complex workflow schemas and complex workflow instances. While each workflow instance can only be a sub-workflow of one complex workflow, workflow schemas can appear as sub-workflows in multiple complex workflows, to support re-use of workflow schemas. In the workflow meta schema, this property is specified as constraint c2. As a result, a set of workflow instance objects which are linked in WF-SubWF Relationships form a tree-like structure. The relationship between a workflow instance and the workflow schema which describes its structure is represented by the SchemaOf-relationship. As is shown by cardinalities of this relationship, each workflow schema can be related to multiple workflow instances, while each workflow instance is related to exactly one workflow schema.

The representation of a complex workflow schema s using the workflow meta schema is now illustrated. Figure 6.4 shows a complex workflow schema s and its sub-workflow schemas 1 to 6, some of which are re-used multiple times in s or its sub-workflow schemas. In the workflow meta schema, each occurrence of a workflow schema as a sub-workflow of a complex workflow schema is represented by a single WF-SubWF Relationship object. Objects of this class are represented by dotted lines, and they are marked $o1, o2, \ldots, o11$, where $o1$ represents the occurrence of workflow schema 6 as a sub-workflow schema of s, $o2$ represents the occurrence of 1 as a sub-workflow schema of s, etc. Notice that the two occurrences of 3 are represented by WF-SubWF Relation-

ship objects $o4$ and $o5$, respectively. The sub-workflows of complex workflow schema 1 are linked to it by WF-SubWF Relationship objects $o8$ and $o9$. These objects are shown twice in Figure 6.4, since both occurrences of 1 reference the same object, and thus, they reference the same sub-workflow objects.

An important feature of a workflow schema is the definition of control flow. Using the information on the workflow activities and on their execution order, a workflow management system can actively control the execution of application processes. Since each workflow schema can be re-used in multiple complex workflow schemas, control flow has to be defined within the context of complex workflow schemas. Hence, in the workflow meta schema, control flow is defined using WF-SubWF Relationships rather than using workflow schema objects. In particular, each control flow is modeled by an ordered pair of WF-SubWF Relationship objects. In a complex workflow schema k with sub-workflow schemas i and j, represented by WF-SubWF Relationship objects $o = (k, i)$ and $o' = (k, j)$, a control flow from sub-workflow schema i to sub-workflow schema j is represented by (o, o'). In Figure 6.4, the occurrences of workflow schema 1 in s are represented by objects $o2$ and $o10$. The control flow constraints of these occurrences are defined using these objects; the control flow constraint of the first occurrence (defined by $o2$) is represented by $(o2, o3)$; the control flow constraint of the second occurrence (defined by $o10$) is represented by $(o10, o11)$.

To represent data flow, each workflow schema is assigned a set of input parameters and a set of output parameters. In the workflow meta schema, the class Parameter has two sub-classes, Input Parameter and Output Parameter, respectively. Each parameter object belongs to exactly one workflow object, and each workflow object can be assigned multiple parameter objects. When a workflow is started, its input parameters are read and used to perform the workflow. When a workflow terminates, its output parameters are written. In general, a data connector links two parameters of workflow schemas. We distinguish between horizontal and vertical data flow. In particular, a horizontal data connector links an output parameter of a sub-workflow with an input parameter of a sub-workflow of the same complex workflow. A vertical data connector models a data flow from a complex workflow to one of its sub-workflows or vice versa. Just like control flow, data flow is defined using WF-SubWF Relationship objects, allowing to define different data flow constraint for different occurrences of the workflow schema.

Since data flow is defined using parameters of workflow schemas, each definition of a data flow involves WF-SubWF Relationship objects and parameter objects. Figure 6.5 shows a complex workflow with horizontal and vertical data flow. In that figure, parameter objects are represented by marked rectangles, where input parameters appear on the left-hand-side, and output parameters appear on the right-hand-side of workflow schemas.

Complex workflow 1 has input parameters x and y and an output parameter z, 3 has an input parameter a and an output parameter b, and 4 has input parameters c and d and an output parameter e. Data flow is represented by dotted arrows between parameters. We begin by discussing the horizontal data flow shown, i.e., the data flow from output parameter b of sub-workflow 3 to input parameter c of 4. This data flow is characterized by the quadruple $(b, o8, c, o9)$. In general, each horizontal data flow is characterized by two WF-SubWF Relationship objects, one output parameter object and one input parameter object. Using WF-SubWF Relationship objects to specify data flow allows to define different data flow constraints for different occurrences of workflow schemas, supporting the re-use of workflow schemas.

In the workflow meta schema, data flow is represented as follows. A class Horizontal Data Flow models the relationship between an output parameter (qualified as source), an input parameter (destination) and two WF-SubWF Relationship objects, qualified as source Workflow and destination Workflow, respectively. Constraint c1a defines that horizontal data flow can only occur between sub-workflows of a common complex workflow and that data flow follows control flow, i.e., that for each horizontal data flow (a, o, b, o'), there is a path of control connectors from o to o'.

Figure 6.5 shows vertical data flow between the complex workflow and its sub-workflows. In the workflow meta schema, vertical data flow is represented by a triple of the form (a, b, o), where a represents the source parameter of the data flow, b the destination parameter and o the WF-SubWF Relationship object. Notice that one WF-SubWF Relationship object suffices to define a vertical data flow, since both workflow objects involved participate in that relationship. For instance, the data flow from parameter x of workflow schema 1 to parameter a of workflow schema 3, were $o8 = (1, 3)$ is the respective WF-SubWF Relationship object, is characterized by the vertical data connector object $(x, a, o8)$. Analogously, $(e, z, o9)$ represents the data flow from sub-workflow schema 4 to s. Notice that in the first vertical data flow, the source and destination parameters are input parameters, while in the second vertical data flow both parameters are output parameters. Hence, in the workflow meta schema, the Vertical Data Flow class relates the Parameter class to the WF-SubWF Relationship class (with qualifiers source and destination), rather than the Input and Output Parameter classes, as the Horizontal Data Flow class does.

The start condition of a workflow is used to evaluate at runtime if the workflow has to be executed. Information passed by data flow can be used for this evaluation. In order to permit different start conditions for different occurrences of a given workflow schema, we bind a start condition to a WF-SubWF Relationship object rather than to a workflow schema object. In the workflow meta schema, the class Start Condition is linked to the

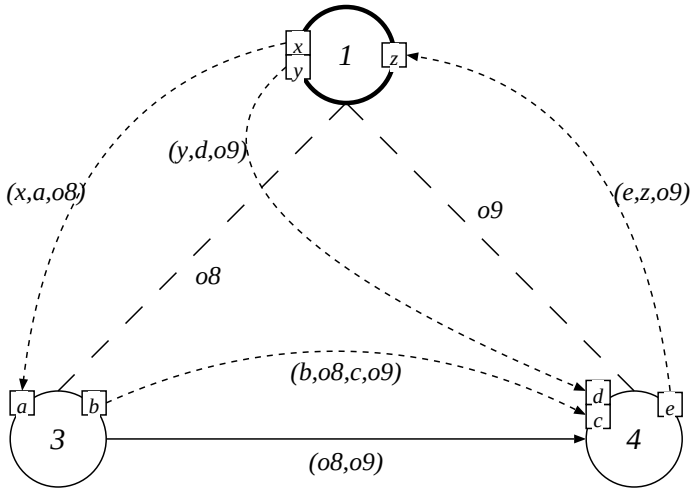

Figure 6.5: Complex Workflow Schema with Parameters and Horizontal and Vertical Data Flow.

WF-SubWF Relationship class and to the Input Parameter class, indicating that each start condition can use a set of input parameters and exactly one WF-SubWF Relationship object. We remark that each start condition defined for a WF-SubWF Relationship object $o = (i, j)$ can only use input parameters, which belong to workflow schema j.

Notice that the workflow meta schema specified by a UML class diagram allows to represent workflow schemas according to the mathematical formalization proposed in Section 3.2. To motivate this fact, Definitions 3.1 and 3.2 have to be reconsidered. In the former definition, workflow schema nodes are defined to consist of input parameters, output parameters, start condition parameters, and a start condition. These artifacts are also represented in the workflow meta schema, i.e., in the Parameter classes and the Start Condition class, respectively. Definition 3.2 specifies the structure of workflow schema graphs. In particular, a workflow schema graph consists of a set of workflow schema nodes, related to each other by control flow and data flow constraints. The workflow schema nodes that belong to the workflow schema graph of complex workflow schema s are represented in the class diagram by WF-SubWF Relationship objects $o = (s, s')$ for each sub-workflow schema s' of complex workflow schema s. It is interesting to observe that analog figures are used to illustrate the definition of data flow in the mathematical formalization (Figure 3.2) and in the object-oriented representation (Figure 6.5), illustrating the close link between the mathematical formalization and the conceptual modeling of workflow schemas. For clarity reasons, start conditions are represented by solid boxes on the left hand side of workflow schema nodes.

After discussing how workflow schemas and their constituents are represented in the

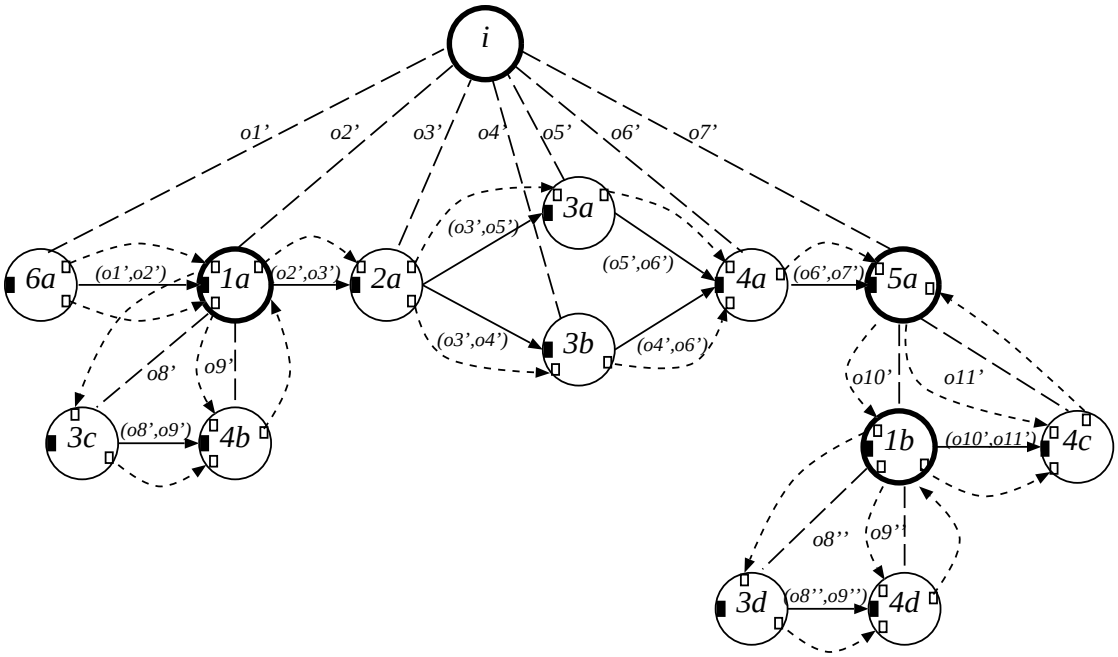

Figure 6.6: Representation of Complex Workflow Instance in Workflow Meta Schema.

workflow meta schema, we now turn to workflow instances, which are also represented in the workflow meta schema. We use a sample workflow instance which is based on the workflow schema shown in Figure 6.4; the workflow instance is shown in Figure 6.6. While in the workflow schema different occurrences of a given workflow schema reference a single workflow schema object, for each occurrence of a workflow schema an individual workflow instance object is created.

In the sample workflow instance shown in Figure 6.6, there are workflow instance objects $3a, 3b, 3c$, and $3d$, based on workflow schema object 3. These are linked to their respective super-workflow by WF-SubWF Relationship objects $o5', o4', o8'$, and $o8''$. Hence, there are objects of the WF-SubWF Relationship class which represent the hierarchical relationship between workflow schemas, and there are objects of that class, which represent the relationship between workflow instance objects. For example, the links between the complex workflow instance i and its sub-workflow instances are represented by WF-SubWF Relationship objects $o1', o2', \ldots, o7'$. Using the SchemaOf-relationship of the workflow meta schema, each workflow instance knows the identify of the workflow schema it is assigned to. As defined by the cardinalities of the workflow meta schema, each workflow instance object has exactly one workflow schema it is assigned to, and each workflow schema can have zero or more workflow instances. Returning to the example, the SchemaOf-relationship contains the objects $(3a, 3)$, $(3b, 3)$, etc.

Figure 6.7 shows complex workflow instance $1a$ including its sub-workflow instances, parameter objects, and data flow objects. Parameter objects belong to workflow instance objects and, thus, are specified by a', b', etc. Consequently, the vertical data flows are represented by vertical data flow objects $(x', a', o8')$ and $(e', z', o9')$; the horizontal data flow shown is represented by the horizontal data flow object $(b', o8', c', o9')$.

We have already observed that the mathematical formalization of workflow schemas complies with the representation of workflow schemas in the class diagram representing the workflow meta schema. This compliance also holds for workflow instances. As specified by Definitions 3.5 and 3.6, workflow instance nodes consist of input, output and start input parameters, and a start condition, all of which have associated values. While the parameters and the start conditions are represented by the Parameter and Start Condition classes, respectively, the types of the parameters are represented in the Data Type class, while the values of the parameters are represented in the Data Object class. According to Definition 3.6, each workflow instance graph consists of a set of workflow instance nodes with control flow and data flow constraints. Just like in the schema level, these constraints are represented by objects of the WF-SubWF Relationship class. The next section discusses the compliance of the object-oriented design and the mathematical formalization of workflow schemas and workflow instances in more detail.

After explaining the central classes of the workflow meta schema, we now turn to the information and organizational aspects. For each parameter, there is a type associated; the types are represented by a class Data Type in the workflow meta schema. Maintaining information on the types of parameters allows for type-checking when building workflow schemas. As indicated by the cardinalities shown in Figure 6.3, each parameter has exactly one type, and each type can have any number of parameters associated. The Data Object class represents the data values of workflow instance parameters. This class maintains the values of application data generated or manipulated in workflow applications. Using the Parameter Instance Reference class, data values are bound to parameters of workflow instances. We remark that modeling application and workflow data and handling it in workflow applications is an important topic in workflow management. Due to the object-oriented approach, the Data Object class and the Data Type class can be used as super-classes to application-specific classes. When application specific business-object classes are available, these can be defined as sub-classes of the Data Object class, in which case the business objects can be used easily in data flow. As will be explained later, the CORBA architecture provides standardized interfaces, which can be used by business objects to be integrated in CORBA-based workflow management systems [21]. The workflow meta schema presented provides methods for an integration of application-specific classes into the novel workflow management system. Integrating application

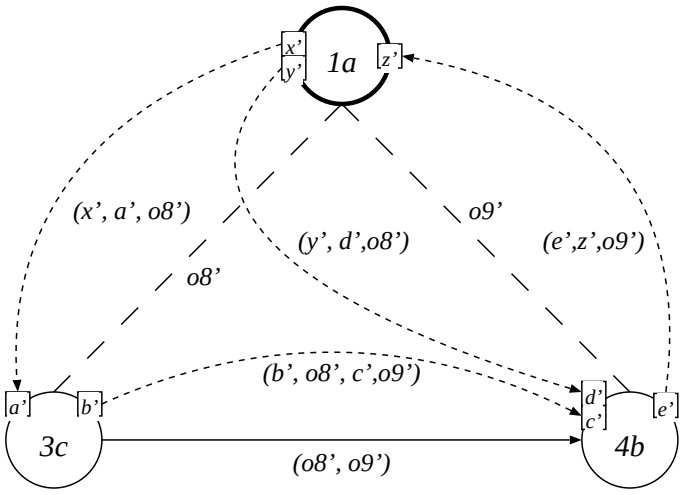

Figure 6.7: Complex Workflow Instance with Parameters and Horizontal and Vertical Data Flow.

specific business objects and their respective classes into the system is discussed when revisiting the design goal in Section 8.4.

The organizational aspect is covered by defining for each workflow schema a role, which is done in the Role class. A role may be a qualification of a person or a software system, needed for the execution of a workflow. At the runtime of a workflow instance, the role information is used to determine an agent, which is capable to perform the workflow instance. Agents are maintained in the Agent class, and each workflow instance is performed by one or more agents. The relationship between agents and roles is maintained in the Role Assignment class, which specifies the roles an agent can fulfill.

6.4 Modeling Dynamic Aspects

Since workflow management systems expose a complex dynamic behavior, to build a workflow management system, an explicit modeling of its behavior is essential. Since in our approach workflow management system functionality is provided by distributed workflow instances, the dynamic behavior of workflow instances has to be described properly. To do so, this section presents a state transition diagram for workflow instance objects, which is based on the one discussed in Section 3.2(cf. Figure 3.11). We remark that each workflow instance object has a state transition diagram implemented; depending on the kind of workflow instance (complex workflow instance, atomic workflow instance), different variants of state transition diagrams are implemented. In the remainder of this section, a generic state transition diagram is presented in some depth, and the specific

properties of the variants are explained.

The dynamic behavior of workflow instances is described using the sample workflow instance shown in Figure 6.6, whose workflow schema is given in Figure 6.4. The creation of the complex workflow instance i is triggered by an application specific event, for instance the arrival of a customer order. For this event, a workflow schema is specified which is used to guide the execution of the corresponding workflow instance; in our case the workflow schema is s. Now the system creates a workflow instance, which is done by creating an object i of the workflow class, where the SchemaOf-relationship defines that the "SchemaOf" i is s. At this point in time, this procedure is performed for each immediate sub-workflow schemas of s, resulting in the creation of workflow instance objects $6a, 1a, 2a, 3a, 3b, 4a, 1b$. These workflow instance objects are related to i by WF-SubWF Relationship objects $o1', o2', \ldots, o7'$, as shown in Figure 6.6. In addition, objects to represent control flow, parameters and data flow are created and linked to the workflow instance objects (as sketched in Figure 6.7 for a sub-workflow instance of i). By analyzing control flow constraints, i determines that $6a$ is the first sub-workflow to be executed (the only one without incoming control connector) and sends a start-message to that workflow object. (If there are more sub-workflows with this property, all of them are started.)

Following the object-oriented approach, objects communicate with each other by sending and receiving messages; in our case, i sends a message, which $6a$ receives and reacts by starting the respective workflow instance. When $6a$ terminates, it fills its output parameter and sends a start-message to the next sub-workflow, i.e. to $1a$. Since $1a$ is a complex workflow, its sub-workflow instances $3c$ and $4b$ are created by that time. Now $1a$ sends a start-message to $3c$, which evaluates it start condition and — if it is evaluated to true — is performed. Role resolution is used to determine the agent skilled and available to perform that workflow activity. When that workflow terminates, it fills its output parameter and sends a start-message to $4b$. Notice that it receives input data from its super-workflow $1a$ and from $3c$. When it terminates, it fills its output parameter. When $4b$ terminates, it sends a termination-message to $1a$, informing it of its completion. Now $1a$ completes and sends a start-message to $2a$. The complex workflow instance continues with the parallel execution of $3a$ and $3b$, followed by the sequential execution of $4a$ and $1b$. When $1b$ terminates, it sends a message to its super-workflow i, informing it of the completion. On receiving this message, i determines if the complex workflow instance is completed.

The specification of the dynamic behavior of workflow instances opens the door for fully distributed workflow executions. In particular, each workflow instance has information on its successor in the workflow schema graph, which is used to send start- and

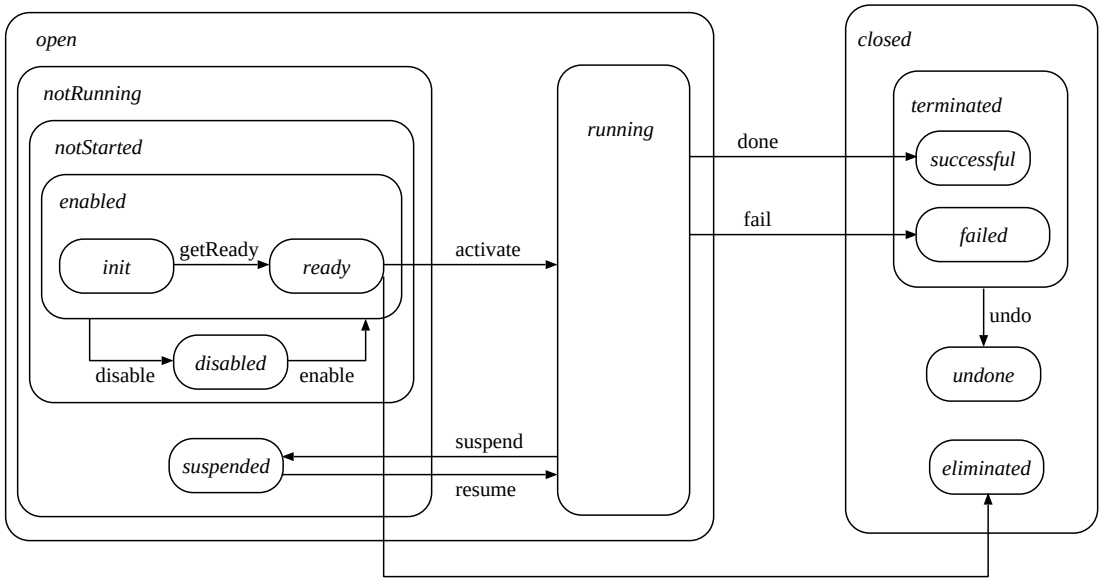

Figure 6.8: State Transition Diagram for Workflow Instance Objects.

termination messages between workflow instance objects. Hence, the workflow instances control their execution autonomously, and a centralized workflow engine is not required. The understanding that workflow instance objects control their execution by sending and receiving messages is in line with the object-oriented approach. In object-oriented systems, each object at each point in time is in a specific state. Objects communicate with each other by sending and receiving messages, and each object reacts on the receipt of a message depending on its current state. The reaction to receiving a message might result in a change of its state. Therefore the dynamic behavior of a system and its objects should be described explicitly using states and state transitions. The resulting model should describe the states an object can reach, and it should specify how an object can change its state, i.e., in which state a message can be received and which state transition occurs. To describe this kind of dynamic behavior, state transition diagrams can be used. In the next paragraphs we will explain the states and state transitions a workflow instance object can walk through during its lifetime.

As sketched in Section 6.3, there are complex workflows and atomic workflows, the latter of which are partitioned into automatic and manual atomic workflows. The general behavior of workflow instances are captured in a generic state transition diagram, which is shown in Figure 6.8. The generic state transition diagram is used as a basis for the specification of the dynamic behavior of the different kinds of workflow instances; some of their specific properties are discussed below.

In Figure 6.8, states of workflow instances are represented by marked rectangles, and

state transitions are specified by marked edges between rectangles. States can be nested, i.e., each state may consist of a number of sub-states and state transitions. We use the dot-notation to refer to nested states. For example, the sub-state *running* in the state *open* can be specified by *open.running*. If the name of a sub-state is evident, we do not use the dot-notation.

When a workflow instance is created, it enters the *init* state. In this state, the preparations for the execution of the workflow instance take place. When the preparations are completed, the state changes via the *getReady* transition to the *ready* state. Using the *disable*- and *enable*-transitions (i.e., calling the respective methods implemented in the workflow class), the workflow instance can be disabled temporarily. Notice that workflow instances can only be disabled before they are started (running workflow instances can be suspended, though). When a workflow instance is in the *ready*-state, it can be activated to change its state to *running*. With the *suspend*-transition, the execution of the workflow instance can be temporarily *suspended*. The suspension can be abolished by the *resume* transition.

The main work of a workflow instance is performed in the *open.running*-state, for instance the execution of an application, a user dialog, or the execution of the sub-workflows of a complex workflow instance. Depending on the success of the execution, the state changes to *closed* via the *fail* or the *done* transition, indicating a failed and successful completion, respectively. As a result, the workflow instance enters either the *successful* or the *failed* state, which are sub-states of the *terminated*-state. Once a workflow instance has reached the *closed*-state, it does not leave it. However, terminated workflow instances can be undone using the *undo*-transition. We remark that the compensation of a workflow instance is not always feasible, but depends on the availability of compensating activities, which undo successfully completed workflows on a semantic level [31]. For instance, a flight reservation can be undone, while a money withdrawal at a teller machine, in general, cannot. Sub-workflow instances whose start condition is evaluated to false are not executed during a particular complex workflow instance. These sub-workflow instances are eliminated, represented by a state transition from the ready state to the *eliminated*-state.

6.5 Compliance with Mathematical Formalization

After introducing the object-oriented design using a class diagram and a behavioral specification using state transition diagrams, the relationship of the object-oriented design to the mathematical formalization of workflow specifications as presented in Section 3.2 is discussed. In the previous section, it was already motivated that workflow schemas

and workflow instances as specified in the mathematical formalization match workflow schemas and workflow instances represented by the object-oriented class diagram. This section takes a broader look on the subject by discussing structural aspects and behavioral aspects in turn. Since mathematics and object modeling obviously aim at different aspects, they are not directly comparable using formal methods. Hence, this section discusses their relationship and motivates their compliance informally, rather than providing formal proof techniques.

6.5.1 Structural Aspects

The structural aspects of the object-oriented design are captured in the workflow meta schema, specified as a UML class diagram in Figure 6.3. In large parts, there is an immediate correspondence between both representations. For instance, complex workflow schemas consist of workflow schemas, which can be atomic or complex. There are control flow and data flow constraints between workflow schemas in the context of a given complex workflow schema. Data flow is partitioned into horizontal and vertical data flow, both kinds of which are based on input and output parameters. In the remainder of this section, the definitions of Section 3.2 are revisited in order to be matched against the object-oriented design.

In Definition 3.1, workflow schema nodes are defined. Each workflow schema node is represented by an object of the workflow class in state schema. As indicated above, workflow objects are separated by their state, such that workflow schema objects are represented by workflow objects in the schema state, while workflow instance objects are workflow objects in the instance state, which in turn is refined by a state transition diagram. The set of typed input parameters of a workflow schema node is represented in the object level by an aggregation between the Workflow class and the Input Parameter class. The typing of parameters is modeled by an association relationship of the Parameter class (as a super class of the Input and Output Parameter classes) and the Data Type class. Analogous considerations show the representation of the set of output parameters in the class diagram. The start condition of a workflow schema is represented by the Start Condition class, and the start condition parameters are represented by an aggregation association to the Input Parameter class.

Control flow and data flow constraints are considered in Definition 3.2, which defines workflow schema graphs. In the class diagram, the set V_s of workflow schema nodes of a workflow schema graph is represented by an indirect recursive relationship between the Workflow Class and the Complex class as one of its sub-classes. This relationship models the fact that each complex workflow schema can have multiple sub-workflow schemas,

each of which is in the Workflow class, i.e., each of which can be either atomic or complex. The occurrence of a workflow schema as a sub-workflow schema of a complex workflow schema is represented explicitly by the WF-SubWF Relationship class. Since the mathematical formalization does not consider technical details, there is no distinction between manual and automatic atomic workflows. This separation, however, is required in an object-oriented design since it is used as a basis for the implementation of a workflow management system.

Since control flow and data flow constraints are present in the context of a complex workflow, these artifacts are attached to the WF-SubWF Relationship class rather than to the workflow class directly, as explained above. Control flow is maintained in a Control Connector class, which relates two WF-SubWF Relationship objects. The class diagram does not model the implementation of control flow by specific parameters, as the mathematical formalization does. This design decision was taken because from a conceptual point of view, the implementation of control flow by control flow parameters is not relevant. This property is included in the mathematical model to properly specify workflow execution semantics. Data flow is represented in the class diagram by a Horizontal Data Connector relationship class, connecting two sub-workflows (represented by WF-SubWF Relationship objects), an input, and an output parameter. Analogous considerations hold for vertical data connectors: They are maintained in a Vertical Data Connector class, which relates an WF-SubWF Relationship object with two parameter objects.

In the context of workflow schema re-use, the mathematical formalization is more restrictive than the conceptual design. In the mathematical model, the sub-workflow schemas of a given complex workflow schema are maintained in a set of workflow schemas. As a result, each workflow schema can occur at most once as a sub-workflow schema of a given complex workflow schema. In the conceptual design, however, multiple occurrences of a given workflow schema in a complex workflow schema may occur. To formalize this situation properly, the mathematical formalization would require multi sets instead of sets, introducing additional complexity. The mathematical formalization does not model the organizational aspect of workflow schemas, i.e., roles and agents are not represented in the mathematical model. Since the formal model specifies formal properties of workflow executions and by stipulation, each workflow instance is performed by a workflow participant selected, the organizational embedding can be omitted from the mathematical formalization. To build a workflow management system, however, it is crucial to model agents and roles involved in workflow executions. In the class diagram, this information is captured in the Agent and Role classes, respectively.

6.5.2 Behavioral Aspects

In the mathematical formalization, the behavior of workflow instances is described by a rather simple state transition diagram, shown in Figure 3.11. Obviously, that diagram is included in the state transition diagram describing the behavior of workflow instance objects, displayed in Figure 6.8. The reason for the higher complexity of the latter state transition diagram is the expected use of the diagram: While the mathematical formalization of workflow executions does not take into account the technical environment and a software architecture, the behavior specification of the object-oriented design has to provide additional states to cope with these issues. This aspect will be illustrated as follows.

An important competence of a workflow administrator is delaying the execution of particular workflow instances or sub-workflow instances. This may be due to specific policies of the organization. For instance, if a customer has open bills, no additional orders are shipped to that customer. Shipping resumes only after the customer has paid his bills. This functionality is represented by an additional state and two state transitions for workflow instances in the state transition diagram shown in Figure 6.8: There is a state *suspended*, which can be entered from the running state by a suspend transition, and there is a resume state transition which brings the workflow instance back to the running state. As will be discussed in the next chapter, there is a technical motivation for suspending workflow instances, which is related to dynamic adaptations of workflow instances. In a system implementation, this functionality can be provided by purging work items from the work item list of workflow participants. Only when the workflow instance can be resumed, the work items are brought back to the work item list, so that the workflow participant can select that item to perform the workflow activity, which resumes the formerly suspended workflow.

6.6 Summary

This chapter presents the conceptual design of a novel workflow management system based on an object-oriented approach and aiming towards workflow schema re-use, flexible workflow execution control, and distributed workflow control. Workflow schemas and workflow instances are represented by objects which reside in a workflow class; a recursive relationship maintains information on which workflow instance objects are based on which workflow schema objects. Workflow schemas can be re-used as sub-workflows in multiple complex workflow schemas, and each of these occurrences can be embedded in a different way, using control flow constraints, data flow constraints, and start conditions, tailored towards the specific occurrences of the particular workflow schema object. The

major design goals are

- flexibility, represented by the ability to adapt running workflow instances to new workflow schemas (to be discussed in detail in the next chapter),
- distribution and scalability, supported by a fully distributed approach to workflow execution control, in which individual workflow instance objects control their execution by message passing,
- persistency, represented in the object design level by objects, which at each point in time are characterized by a state, which can be stored persistently,
- re-use of workflow schemas, supported by representing occurrences of workflow schema objects in multiple complex workflow schema objects by specific objects of the WF-SubWF Relationship class,
- integration, represented by including in the workflow meta schema data type and data object classes, such that the elements of the latter can be used in workflow schemas to specify parameters and data flow between sibling sub-workflows.

The dynamic behavior of workflow instances is specified by state transition diagrams. While a state transition diagram has been used in the mathematical formalization in Part I, it has to be refined in this chapter to capture technical requirements which are necessary for the conceptual design of a workflow management system. Finally, the relationship of the mathematical formalization as presented in Part I to the object-oriented design of a novel workflow management system as presented in this chapter, are discussed. In particular, it is motivated that the mathematical formalization complies with the object-oriented design in the sense that it provides formal properties, while it abstracts from technical aspects, which are, however, important for the design and implementation of a complex software system.

Chapter 7

Dynamic Adaptations

Based on the graph-based workflow language formally specified in Chapter 3 and on the corresponding workflow meta schema introduced in the previous chapter, dynamic adaptations of running workflow instances are addressed.

There are numerous questions which have to be solved in the context of dynamic adaptations, some important ones of which are given in the following list:

- How can dynamic adaptations of running workflow instances be designed and implemented in a workflow management system?
- How are dynamic adaptations controlled, and what correctness criteria are suitable? In particular, which rules govern whether a workflow instance can be adapted to a new workflow schema?
- What is the scope of a dynamic adaptation, i.e., which workflow instances should be adapted to a new workflow schema?
- Who is permitted to perform dynamic changes, and under which conditions?

In the remainder of this chapter, issues related to the first, second, and third item will be investigated. Organizational issues of dynamic adaptation will not be covered, since these issues are more related to the business domain of organizational modeling that to technical issues of advanced workflow management. In any case, dynamic adaptations will have to be embedded in an organizational framework, which specifies who is allowed to change the structure of workflow instances, and what is the scope of these dynamic adaptations. This thesis concentrates on the technological aspects of controlled dynamic adaptations by presenting the conceptual design and prototypical implementation of a workflow management system which is capable of controlled dynamic adaptations. Using

the role mechanism discussed earlier, however, the organizational policies with respect to the organizational embedding of dynamic adaptations as defined by business domain experts can be implemented in the system.

This chapter is organized as follows: Workflow application development processes are revisited, and the implications of dynamic adaptations on workflow application development processes is investigated. Refining the considerations in Section 5.2.1, Section 7.2 reconsiders the build-time versus run-time approach and the interpretation-based approach to workflow execution control. Since it is not feasible to allow arbitrary dynamic adaptations, they have to be controlled by workflow management systems. Correctness criteria which are used to decide whether a given workflow instance can be adapted to a new workflow schema are presented in Section 7.3.

7.1 Dynamic Adaptations and Workflow Methodology

The design methodology for workflow applications presented in Chapter 4 provides a framework to assist participants in workflow application development projects in planning and conducting these projects. As sketched above, the design methodology can be tailored according to the needs of specific workflow projects. The remainder of this section sketches how the design methodology can be tailored towards adaptive workflows. Two examples are discussed, namely installing a new external application to be used by running workflow instances and structural dynamic adaptations, i.e., changing the structure of a set of running workflow instances to comply with a new workflow schema.

For workflow instances to make use of new and presumably more efficient external application programs, the installation of a new external application has to be embedded in the context of the workflow application development methodology. If workflows are long running and there are numerous active workflow instances of a given workflow schema, the organization can benefit considerably from changing an old external application with a new and better one. Of course, the organization wants as many workflow instances as possible to make use of the new tool. With respect to workflow management, this can be considered a modification of running workflow instances to changes in the environment, more precisely the technical environment of the application process. It is assumed that currently there is already a workflow application running and the workflow management system is capable of providing the functionality required for changing the assignment of an external application to a workflow schema and transfer this assignment to all running workflow instances based on that schema.

This scenario is captured by the design methodology as follows. First, the new exter-

nal application is entered into the system, which is reflected by activities in the Design phase, shown in Figure 4.3. In particular, information on the new computational infrastructure is entered into the system, using the application definition sub-phase of the Design phase. Since there are no changes to the other aspects of the workflow schema, the workflow modeling sub-phases can be omitted. Now the Implementation phase is entered (cf. Figure 4.4), in which the interface of the new external application is developed, which is required for the integration of the new external application into the workflow application. This activity is performed in the tool integration sub-phase of the Implementation phase, followed by extensive testing and review of the integration to make sure the new external application can be used safely in the workflow application.

Once the decision is taken to use the new external application in all future workflow instances and also in running workflow instances, the operational aspects of the workflow schemas involved are changed to use the new external application. As specified in workflow schemas, each running workflow instance involved is now changed to use the new application. While the technical details like data flow compatibility of the original tool integration and the new tool integration are left to workflow projects, the embedding of the activities required for changing an external application program in the context of the workflow application design methodology assists in adapting running workflow instances to changes in the technological environment of the application process.

In a second, more advanced example, the structure of a complex workflow schema is changed. Information on suitable improvements of workflow schemas can be derived from execution information, which is gathered during the Operational phase, shown in Figure 4.6. The new structure of the workflow is developed during the Design phase of the workflow methodology. Notice that the modifications are initially specified as business process models, not as workflow schemas. Since there is already an operational workflow application, the System Selection phase can be omitted. Turning to the Implementation phase, the changes have to be reflected on the workflow level. Hence, the improvements as specified on the level of business process models have to be implemented in a workflow schema. Typical activities to perform on an original workflow schema s to develop a new workflow schema s' are adding new sub-workflows and deleting others, or adding or deleting execution constraints, like control flow or data flow. After testing the new workflow schema for correctness, it is ready for usage in future workflow instances and in running workflow instances. While the former is trivial — new workflow instances just use the new workflow schema —, the latter amounts to dynamic adaptations of running workflow instances, which requires some additional remarks.

Just like in the first example where a new and presumably better external application has to be made available for all future and as many active workflow instances as possible,

the new workflow schema shall also be used by all active workflow instances for which the adaptation is meaningful. However, there is a considerable difference between installing and using a new external application and installing and using a new workflow schema. While installing a new external application has implications only to atomic workflows, structural changes to complex workflow schemas may involve all sub-workflow schemas of the changed complex workflow schema as well as their control flow and data flow constraints. Therefore, for each workflow instance the workflow management system has to check whether it can be adapted to the new workflow schema.

In general, there are numerous workflow instances based on a given workflow schema, all of which may be in a different state of execution: Some may have just started, others may be in the middle of execution, while still others may already be about to terminate. Depending on the state of execution, a given workflow instance may or may not make use of the new workflow schema. The task of the workflow management system is to select the workflow instances which can use the new workflow schema, i.e., which are adaptable to the new workflow schema. There are consistency criteria which have to be evaluated for each running workflow instance, once it faces adaptation to a new workflow schema. The correctness criteria are discussed below. While in general not all workflow instances can be adapted to a new schema, the original workflow schema is kept, which can then be used to control the execution of the running workflow instance which cannot or which need not be adapted to the new workflow schema. In particular, a workflow instance for which the adaptation is correct will resume execution with the new workflow schema, while a workflow schema for which the adaptation cannot be allowed will continue with the original workflow schema. In addition, the workflow administrator can choose to terminate workflow instances, for which from a semantic point of view a continuation with the original workflow schema cannot be permitted.

Before going into the details of dynamic adaption correctness and implementation, two general approaches to workflow execution control are discussed, one of which is not adequate for dynamically adapting workflow instances, while the other is.

7.2 Workflow Execution Control

From a system's point of view, there are different approaches to modeling and controlling the execution of workflows. Traditionally, a build-time versus run-time approach is used, characterized by a strict separation of workflow modeling (build time) and workflow execution control (run time). In addition, the relationship of a workflow instance to its workflow schema is static and cannot be changed during workflow executions. In a

second approach, workflow schemas are interpreted at run time, and the assignment of a workflow schema to a workflow instance may change dynamically, i.e., while the workflow instance runs. These approaches are discussed in turn, and their implications to the design and implementation of dynamic adaptations in workflow management systems are investigated.

7.2.1 Build-Time versus Run-Time Approach

In the build-time versus run-time approach, there is a strict separation of workflow modeling and workflow execution control. During the build time of a workflow, a workflow schema is specified completely, typically using a graphical workflow modeling tool which is the component of a workflow management system used for specifying workflow schemas. When a workflow schema is created which satisfies the requirements imposed by the application process, workflow modeling is completed. Depending on the workflow management system used, the workflow schema is represented by a script, written in the workflow language of that system. Workflow schemas can also be stored in a database or workflow schema repository. In any case, the workflow modeling tool is exited after workflow modeling is completed, i.e., the build time of the workflow is completed.

When an application process starts which is supported by a workflow management system, a workflow instance is created. In order to do so, the run time tool of the workflow management system is launched, and a workflow instance is created based on a predefined workflow schema. In a next step, the workflow instance is started. Workflow instances typically live in main memory of the program which controls the execution of workflow instances. This program is called workflow engine. The workflow engine decides for a given complex workflow instance which sub-workflows can be started, and it communicates with workflow clients, which in turn are accessed by workflow participants. In this traditional workflow scenario, there is no link between a workflow instance and its workflow schema after the workflow instance has started. This property implies that changing the workflow schema does not affect running workflow instances based on that workflow schema.

The situation described is somehow similar to traditional programming, where a program is coded in a programming language (build time), compiled to executable code, and executed (run time). In the build-time versus run-time approach, workflow modeling can be regarded as a form of high-level programming, such that the workflow schema represents a program, written in a workflow language. To execute such a workflow program, the run time environment of the workflow management system is used. As a result, workflow instances correspond to program executions. When a workflow instance is created

and its execution is started, the link of the workflow program to the workflow execution is no longer present. In traditional programming there is no way to change the program in execution by changing the program code.

We now discuss if and how a dynamic adaptation of a running workflow instance to a new workflow schema can be done by a workflow management system, which is based on the build-time versus run-time approach. Assume there is a workflow schema s and a workflow instance i, using s. Assume workflow instance i has started, when there is a change in the environment of the workflow, resulting in the need to change i dynamically to reflect that change. Since i is not able to perform any useful work given the new situation, it is stopped. Now there are two options:

1. The work to resolve the new situation can be performed without assistance from a workflow management system. In this case, the workflow participant — the domain expert — disregards the workflow management system and uses traditional techniques and tools to perform the work.

2. Since a change during the run time of the workflow is not feasible in this workflow execution approach, the build time component of the workflow management system is started, and a new workflow schema is created, which describes the structure of the changed application process. Now a new workflow instance i' is instantiated, using the new workflow schema.

From a workflow perspective, the first alternative is not desirable, since there is no workflow support and the workflow management system looses its ability to document application processes. The second alternative, however, has a number of drawbacks, too. The new workflow instance i' has to "inherit" the execution environment of the stopped workflow instance. In particular, data generated or modified during the stopped workflow instance i has to made available to i', which is not an easy task from an implementation point of view. In addition, the new workflow instance i' does not have to re-execute subworkflow instances which have already been performed during i. Hence, i' has to begin its execution at the point at which i was stopped. Other issues remain, e.g., what happens to the other active workflow instances based on the modified workflow schema s?

What makes this situation even more unattractive — and in fact impractical — is the fact that (depending on the implementation of the workflow management system) not only the workflow instance under consideration has to be stopped, but the run-time environment of the workflow management system has to be stopped altogether. Typically there are numerous workflow instances running at that point in time, some of which control the execution of mission-critical processes of an organization. As a result of a dynamic

adaptation of a single workflow instance, all other workflow instances may have to be stopped, too. Clearly, stopping mission-critical applications is not a valid option in commercial applications.

As a result, the alternatives provided by workflow management systems based on the build-time versus run-time approach face severe drawbacks when it comes to dynamic adaptations. Abandoning the built-time versus run-time approach in favor of an interpretation-based approach is adequate in this context, as the following considerations show.

7.2.2 Interpretation-Based Approach

In the interpretation-based approach, there is no strict separation of a workflow's build time and its run time. In contrast, a workflow instance is controlled by interpreting a workflow schema. Hence, there can be alternations between workflow modeling phases and workflow execution phases for a given workflow instance. In these alternations, the workflow schema of a running workflow instance can be augmented, and due to the interpretation-based approach used, these changes immediately effect the workflow instance under consideration. However, the changes only effect it, if parts of the workflow schema are changed, which are not already completed. If parts are changed which already have been completed by the workflow instance then the changes to the workflow schema have no effect on the workflow instance. These considerations lead to consistency criteria for dynamic adaptations, discussed later in this chapter. The interpretation-based approach allows a flexible assignment of workflow instances to workflow schemas, which may even be subject to changes. This means that during different points in time, a workflow instance can be controlled by different workflow schemas. However, at each point in time, each workflow instance is assigned a single workflow schema.

In order to discuss the interpretation-based approach to workflow execution control in more detail, a sample workflow is used. A workflow instance i based on a workflow schema s is created, and its execution starts. Up to this point in time, there is no difference between the interpretation-based approach and the build-time versus run-time approach, discussed above. Now the workflow administrator decides to change workflow instance i. In order to do so, a new workflow schema s' is created, which reflects the required adaptations. In terms of the build-time versus run-time approach, the run time of workflow instance i is interrupted by a build-time phase (or, more precisely, a re-build time phase). When this phase is completed, the assignment of workflow instance i to its workflow schema is changed from s to s' to reflect the dynamic adaptation. When this re-assignment is completed, the execution of the workflow instance i continues with the new workflow

schema s'. It is obvious that changing of the assignment of a workflow instance and a workflow schema is not feasible in the build-time versus run-time approach.

An additional question arises: What happens to the other workflow instances which are controlled by the original workflow schema s? They can execute as if no change had occurred, or they can also be adapted to the new workflow schema s', if the workflow administrator decides to do so and if certain criteria — which are discussed below — are met. This high degree of flexibility can be provided by a versioning mechanism: Whenever there is a change to an existing workflow schema, the original workflow schema is retained and a new workflow schema is created. We adopt this strategy since there are typically additional workflow instances based on the original workflow schema s. As will be discussed shortly, the system provides mechanisms which allow to decide which workflow instances can be adapted to a new workflow schema, and it will assist the workflow administrator in defining which workflow instances will actually be affected by the adaptation.

With regard to dynamic adaptations, there are the following major differences of the build-time versus run-time and the interpretation-based approach:

1. In the interpretation-based approach, a flexible assignment of workflow schemas and workflow instances is feasible. When a workflow instance starts and up to a dynamic adaptation, it is controlled by workflow schema s — after it is adapted, it is controlled by workflow schema s'.

2. Instead of two workflow instances i and i', there is a single workflow instance i. Hence, issues related to the inheritance of the workflow environment and intermediate start of new workflow instances do not occur in this approach.

3. The workflow management system's normal operation is not affected by a dynamic adaptation, and other workflow instances can proceed normally.

7.3 Correctness Considerations

As was indicated earlier, the correctness of workflow instances is defined by the goals of the application process whose execution is supported by the workflow management system. In particular, correctness properties of workflow instances are governed by the application process and, consequently, are specified in workflow schemas. Recall that workflow schemas are formal representations of the automated parts of application processes which are designed to meet the goals of the process under consideration. A workflow instance is therefore correct if it satisfies the constraints imposed by the workflow

schema. Examples of these constraints are control flow, data flow, and roles used to specify persons who perform workflow activities. This general notion of correctness in workflow systems is rather informal, yet adequate, since the correctness is defined by the domain experts in business process models and — in the workflow side — in workflow schemas. As a result, formal and generic correctness criteria known from database transaction processing, like transaction serializability or recoverability, are not adequate to specify application-specific correctness properties in the workflow context.

Since the correctness of workflows is specified by workflow schemas and workflow instances are the automated parts of application processes, the workflow management system has to guarantee that the workflow instance satisfies the criteria defined by the respective workflow schema. In addition, there are correctness considerations in the context of dynamic adaptations. They will be in the center of this section. Recall that consistency properties of workflow instance have already been defined in the context of the mathematical formalization of workflow schemas and workflow instances in Section 3.2 (cf. Definitions 3.5 and 3.6). In these definitions, the structure of workflow instance nodes and of workflow instance graphs are defined. In particular, a workflow instance node is defined based on properties of a particular workflow schema node, and the definition of a workflow instance graph is based on a workflow schema graph.

These definitions are valuable for correctness considerations for dynamic adaptations, since they govern whether a given workflow instance is correct with respect to a given workflow schema. In particular, a complex workflow instance i is correct with respect to a workflow schema s, if Definition 3.6 is satisfied. As will be explained below, these definitions serve as a basis for deciding whether a dynamic adaptation is correct, i.e., whether a given workflow instance can be adapted to a new workflow schema. Before formalizing the concepts, an example of a dynamic adaptation shows the rational behind our considerations.

Consider workflow schema s shown in Figure 7.1(a) and a modified workflow schema s' in part (b) of that figure. As can be seen, the modified workflow schema has additional sub-workflow schemas. In terms of control flow constraints between the sub-workflow schemas, the new sub-workflow schemas are inserted before sub-workflow schema 4. Although that figure abstracts from parameters and start conditions, the overall structure of the workflow schema and of the workflow instance suffice to present the general idea of workflow instance correctness in presence of dynamic adaptations. (The formal correctness criteria will also consider data flow constraints and start conditions.) In Figure 7.1(c), a workflow instance based on workflow schema s is shown. We use a shading to indicate sub-workflow instances which have already started, i.e., which are not in the notStarted state. Hence, all sub-workflow instances except for the last sub-workflow instance of i

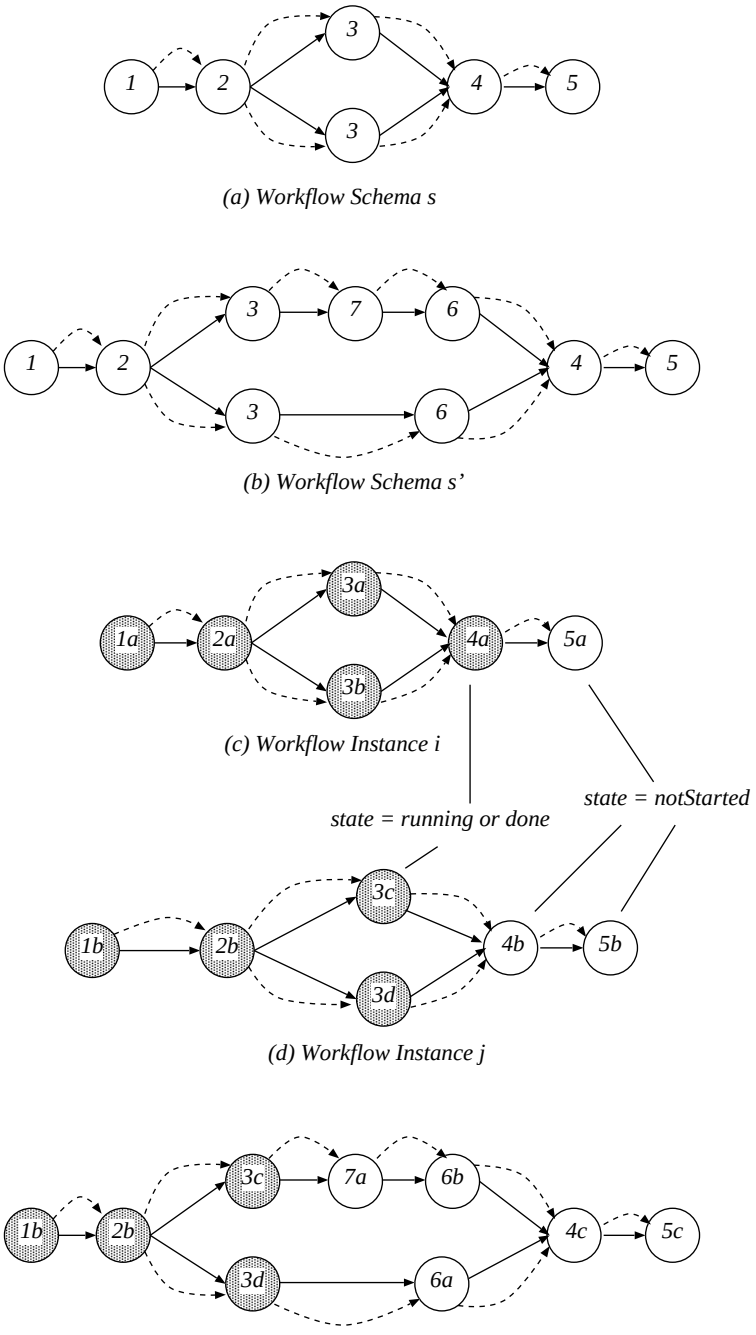

Figure 7.1: Sample Workflow Schemas (a), (b), Workflow Instances (c), (d) and Dynamically Adapted Workflow Instance (e).

have already started. Due to the control flow constraints of workflow schema s, at least workflow instances $1a$, $2a$, $3a$ and $3b$ have already terminated, but all that matters for dynamic adaptations is that they have already started.

A criterion for deciding whether a complex workflow can be adapted to a new workflow schema can be defined as follows:

Definition 7.1 A complex workflow instance i *is adaptable* to a workflow schema s' if and only if there is a continuation of i such that i complies with workflow schema s'. $\diamond$

This definition of workflow instance adaptability is used to check whether workflow instance i as shown in Figure 7.1(c) can be adapted to workflow schema. This is done by testing if i can be continued in a way that a workflow instance results, which matches the new workflow schema s'. In the workflow instance shown, the adaptation is not possible, since sub-workflow instance $4a$ has already started, as indicated by the shading of that workflow instance node. To comply with the new workflow schema s', additional sub-workflow instances have to be executed *before* $4a$ can start, namely a sub-workflow instance for sub-workflow schema 7 and two workflow instances based on workflow schema 6. Hence, there is no continuation for the complex workflow instance that satisfies the control flow constraints imposed by the new workflow schema. Consequently, i is not adaptable to s'.

Workflow instance j shown in Figure 7.1(d) is investigated next. As shown in the workflow instance graph, sub-workflow instance $4b$ has not yet started. Since the sub-workflow instances already started comply with the ones in the new workflow schema s', there is a continuation of j that is consistent with s'. After the check is performed, new sub-workflow instances can be created, as well as their embedding in the context of the complex workflow. After the sub-workflow instances are created, the execution of the dynamically adapted workflow instance can resume. The resulting workflow instance is shown in Figure 7.1(e).

From a conceptual point of view, it is feasible to retain the sub-workflow instances which are also defined in the new workflow schema, $4b$ and $5b$ in the example. However, it is also valid to delete all sub-workflow instances which have not yet been started and create new workflow instances based on the new workflow schema. In the example, the second option was used, as can be seen by the different workflow instance identifiers ($4b$ versus $4c$ and $5b$ versus $5c$) in parts (d) and (e) of Figure 7.1. Notice that the sub-workflow instances which have already been executed must, of course, be identical, for instance $1b$ is the first sub-workflow instance of complex workflow instance j in parts (d) and (e) of that figure.

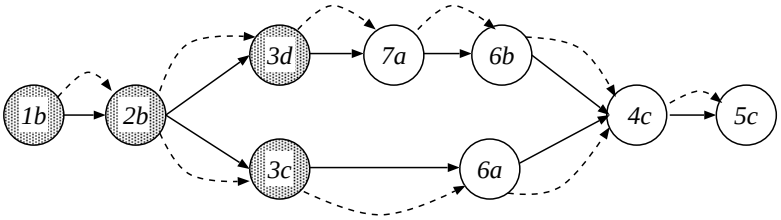

Figure 7.2: Alternative Mapping in Dynamic Adaptation.

To put these considerations more formally, the decision whether or not a workflow instance can be adapted to a new workflow schema is based on mappings. In particular, the sub-workflow instances which have already been started are mapped to the sub-workflow schemas of the new complex workflow schema. In addition, control flow and data flow constraints also have to be taken into account. In our example, sub-workflow instance $1b$ of workflow instance j is mapped to sub-workflow schema 1 of workflow schema s', and sub-workflow instance $2b$ is mapped to sub-workflow schema 2 of s'. The two sub-workflow instances $3c$ and $3d$ are mapped to the occurrences of workflow schema 3 in s'. Since furthermore the control flow and data flow constraints of the workflow instance and the workflow schema comply, a mapping can be found. Hence, j is adaptable to s'.

We remark that in fact there are two mappings with this property. This is due to the fact that the sub-workflow instances $3c$ and $3d$ can be mapped arbitrarily to the two occurrences of sub-workflow schema 3 in workflow schema s'. In effect, the workflow instance can be dynamically adapted and continued in two ways, depending on the mapping selected. In the first choice, workflow instance $7a$ is executed after $3c$, and workflow instance $6a$ is executed after $3d$. In the second alternative, $7a$ is executed after $3d$, while $6a$ is executed immediately after $3c$. Notice that the graphical layout of the sub-workflow schemas is not relevant for the mapping; it is just the workflow schema that matters.

While the workflow management system can compute the set of possible mappings for a given dynamic adaptation, the decision on which mapping to choose has to be taken by the person responsible for the dynamic adaptation, based on application specific knowledge and the aim of the dynamic adaptation. The workflow instance resulting from the mapping just described is depicted in Figure 7.2. The different mappings have different meanings from an application point of view. While in the first alternative, sub-workflow instance $7a$ receives data from workflow instance $3c$ and $6a$ from $3d$, in the second alternative the data flow is between $3d$ and $7a$ and between $3c$ and $6a$.

After introducing the general idea of dynamic adaptations, a more detailed view on the subject is taken. In general, a dynamic adaptation of a workflow instance i to a workflow schema s' is performed in the following steps:

1. *Suspension of workflow instance i:* To suspend i, all sub-workflow instances of i which are in the running state are completed, but no additional sub-workflow instances are started.

2. *Checking whether adaptation of i to s' is possible:* Using formal correctness criteria (which are introduced below), the workflow management system checks whether a dynamic adaptation of workflow instance i to workflow schema s' is possible. In case multiple adaptations are possible (as shown in the example above), the person responsible for the dynamic adaptation selects one alternative.

3. *Perform the adaptation:* This step involves the deletion of obsolete sub-workflow instances of workflow instance i and typically the creation of new sub-workflow instances as well as accompanying data flow and control flow connectors, as specified in workflow schema s'.

During the adaptation process, obsolete workflow instance nodes and obsolete control flow and data flow connectors need to be purged from a workflow instance graph. In particular, all sub-workflow instances which have not yet been started need to be purged, since they do not need to be taken into account during the adaptation process.

Definition 7.2 Given a workflow instance i and a set $O_i \subseteq V_i$ of obsolete workflow instance nodes of i, such that

$$O_i := \{j \mid j \in V_i \wedge \text{StateOf}(j) = \text{notStarted}\}$$

the *purged workflow instance graph of* i is defined by

- Purge from V_i all obsolete workflow instance nodes:

$$V_i := V_i - O_i$$

- Purge all control flow connectors and data flow connectors from C_i and D_i with a source or target node $j \in O_i$.

$\diamond$

In the formal foundation presented in Chapter 3, the assumption was made that each workflow schema can occur only once as a sub-workflow schema of a given complex workflow schema. This limitation was lifted in the object-oriented design of a workflow management system, as developed in Chapter 6. Now that formal techniques are used

to specify the correctness of dynamic adaptations, we again assume singularity of sub-workflow schema objects in the context of a given complex workflow schema. Without loss of generality, this singularity can be achieved by implicitly considering WF-SubWF Relationship objects instead of workflow schema objects, since they do provide singularity. However, in order to keep the presentation clear, we still use workflow schema objects and assume singularity of workflow schema objects, bearing in mind that no restriction on the generality of the results are imposed by that stipulation.

Definition 7.3 A function $m : V_i \mapsto V_s$ is called *a mapping from workflow instance i to workflow schema s* if for each workflow instance node $j' \in V_i$ there is a workflow schema node $j \in V_s$ such that SchemaOf$(j') = j$, i.e.,

$$m(j') = j \Rightarrow \text{ SchemaOf}(j') = j$$

and each workflow instance node is mapped to one and only one workflow schema node, defined by

$$m(j') = m(k') \Rightarrow j' = k'$$

for $j', k' \in V_i$. ◇

Disregarding dynamic adaptations for the time being, we remark that for a workflow instance i, whose execution is controlled by a workflow schema s, a mapping from i to s exists. This observation is based on the fact that a complex workflow instance based on a given complex workflow schema is initiated by creating for each sub-workflow schema of the complex workflow schema a sub-workflow instance, and control flow and data flow constraints between them.

The argumentation continues as follows: Based on a mapping from the purged workflow instance i, valid mappings are defined. A mapping is valid if the execution constraints found in the workflow schema are not contradicted by the workflow instance, and vice versa. This requires to look at execution order constraints specified by control connectors as well as data constraints defined by data connectors. In particular, for each control connector in the workflow schema s' connecting workflow schema nodes which have corresponding workflow instance nodes in i, there has to be a corresponding control connector in the workflow instance. If there is no such control connector then the respective workflow instances are either executed concurrently or there is an additional workflow instance which is performed in between them. This situation has to be ruled out for valid mappings. Regarding properties of valid mappings with respect to data flow, for each data connector in the workflow instance there has to be a corresponding data connector in the workflow schema, both with respect to horizontal and vertical data flow.

Definition 7.4 Given a purged workflow instance i, a mapping m from i to s' is a *valid mapping*, if the following conditions hold:

- For each control connector in s' connecting two sub-workflow schemas which have corresponding sub-workflow instances in V_i, there is a control connector in C_i, and vice versa: $\forall (p,q) \in C_{s'}$, such that $p \in out_k,\ q \in sip_l$ and there exists $k', l' \in V_i$, such that $m(k') = k \wedge m(l') = l$, there is a control connector $(p', q') \in C_i$, such that $p' \in out_{k'},\ q' \in sip_{l'}$.

  Analogously, for each control connector $(p', q') \in C_i$, such that $p' \in out_{k'},\ q' \in sip_{l'}$ and $k', l' \in V_i$, such that $m(k') = k \wedge m(l') = l$, there is a control connector $(p, q) \in C_{s'}$, such that $p \in out_k,\ q \in sip_l$.

- For each data connector in workflow instance i, there is a data connector in workflow schema s', specified by: $\forall (p', q') \in D_i$, such that $p' \in out_{k'}$ and $q' \in in_{l'} \cup sip_{l'}$ and $k', l' \in V_i$, there is a corresponding data connector $(p, q) \in D_{s'}$, such that $p \in out_k$ and $q \in in_l \cup sip_l$ between workflow schemas $k, l \in V_{s'}$, such that $m(k') = k \wedge m(l') = l$.

  Analogously, for each data connector in the workflow schema connecting two sub-workflow schemas which have corresponding sub-workflow instances in V_i, there is a corresponding data connector in the workflow instance: $\forall (p, q) \in D_{s'}$, such that $p \in out_k$ and $q \in in_l \cup sip_l$ between workflow schemas k and l, such that there exist $k', l' \in V_i$ with $m(k') = k \wedge m(l') = l$, there is a data connector $(p', q') \in D_i$, such that $p' \in out_{k'}$ and $q' \in in_{l'} \cup sip_{l'}$.

$\diamond$

Given the definition of valid mappings, the following consideration is valid regarding the execution of a complex workflow instance: If a workflow instance in a given state has a valid mapping with respect to a given workflow schema then there are valid mappings for that workflow instance in all states since its initiation. This observation is formalized in the following lemma:

Lemma 7.1 If a workflow instance i has a valid mapping with respect to a workflow schema s' then i has valid mappings with respect to s' in all previous states since its initiation.

The proof of this lemma is based on the observation that the existence of a valid mapping depends on the purged workflow instance graph of i. It can easily be seen that in earlier stages of workflow instance i, more workflow instance nodes can be purged from G_i than

in later stages. If there is a valid mapping of i with respect to s' in later stages then obviously there is a mapping of i with respect to s' in earlier stages, proving the claim. Given the definition of a valid mapping, it is now obvious how adaptability of workflow instances is defined: by the presence of a valid mapping between a workflow instance and the new workflow schema the workflow instance should be adapted to.

Theorem 7.1 A workflow instance i can be adapted to a workflow schema s' if there is a valid mapping m of i with respect to s'.

Proof This theorem is proved by construction. In particular based on the existence of a valid mapping, the purged workflow instance graph is enhanced to become a workflow instance graph which satisfies the requirements of workflow schema s', as specified in Definition 3.6.

Since a valid mapping of i with respect to s' exists, a prefix of the workflow schema s' is already instantiated and performed by workflow instance i. In particular, all sub-workflow instances in V_i are executed, and need not be created. In order for i to become a workflow instance which complies to s', additional sub-workflow instance nodes and corresponding control flow and data flow constraints have to be instantiated:

- For each sub-workflow schema j in $V_{s'}$ for which there is no corresponding node in V_i, i.e., for which no $j' \in V_i$ with $m(j') = j$ exists, a workflow instance node j' is created and added to V_i, such that SchemaOf$(j') = j$.

- For each control connector $(p, q) \in C_{s'}$, such that $p \in out_j, q \in sip_k$ which has as a source or target a workflow schema node, for which in the previous step a workflow instance node was created and which therefore has no corresponding control connector in C_i, the respective control connector $(p', q') \in C_i$ is created.

- For each data connector $(p, q) \in D_{s'}$ which has as a source or target a workflow schema node for which in the first step a workflow instance node was created (and which therefore has no corresponding data connector in D_i), the respective data connector $(p', q') \in D_i$ is created.

After these operations, $G_i = (V_i, C_i, D_i)$ complies to workflow schema graph $G_{s'}$ as specified in Definition 3.6, proving the claim. □

Relationships of the original workflow schema and the new workflow schema have implications on the feasibility of dynamic adaptations. For instance, if the new workflow schema is created by adding sub-workflows to the original workflow schema and the added sub-workflows are executed after the original sub-workflows then the new version

can be seen as a continuation of the original workflow schema. In this case, all workflow instances can be adapted to the new schema version. If on the other hand a new sub-workflow schema is introduced, which is the first to be executed, then no running workflow instance can be adapted to the new workflow schema.

In particular, if from the properties of the original workflow schema and the new workflow schema it is obvious that no adaptations are possible then no mappings have to be calculated. If on the other hand all workflow instances can be adapted then the mapping has to be calculated for those workflow instances, which actually are being changed. These considerations can be used to optimize the consistency check during dynamic adaptations. These considerations can be formalized using the definition of prefixes of workflow schemas. In particular, if the original workflow schema of the workflow instance under consideration is a prefix of the new workflow schema then a valid mapping exists. This property is formalized in the following theorem:

Theorem 7.2 A workflow instance i with workflow schema s can be adapted to a workflow schema s' if s is a prefix of s'.

Proof Due to Theorem 7.1, it suffices to show that if workflow instance i is based on workflow schema s and s is a prefix of s' then a valid mapping from i with respect to s' exists. As specified in Definition 3.4, s is a prefix of s' if and only if G_s is a sub-graph of $G_{s'}$ and if there is no control flow or data flow connector from a workflow schema node in the postfix (i.e., in $V_{s'} - V_s$) to a workflow schema node in the prefix V_s. According to Lemma 7.1, if a valid mapping of i with respect to s' exists then in each previous state during the execution of i, a valid mapping with respect to s' exists. Hence it suffices to consider a situation, in which all sub-workflow instances of i have already been started.

In this situation, for each workflow instance node $j' \in V_i$ there is a corresponding workflow schema node $j \in V_s$, such that SchemaOf$(j') = j$. Since s is a prefix of s', j is also contained in $G_{s'}$, i.e., $j \in V_{s'}$. Hence, $m : V_i \mapsto V_{s'}$ is a mapping, such that $m(j') = j$ for all $j' \in V_i$, such that $j \in V_{s'}$. Due to the fact that s is a prefix of s' the mapping m is a valid mapping, since the control flow constraints and the data flow constraints as specified in workflow schema s are also present in workflow schema s'. □

So far, just a single workflow instance was considered for dynamic adaptation. However, in general there are multiple workflow instances based on a given workflow schema, and all of these workflow instances should make use of a new and presumably better workflow schema. In this case, the check based on valid mappings will be conducted for each workflow instance based on the original workflow schema. The workflow administrator will then be presented a list of workflow instances for which the consistency check was

successful, i.e., which can be adapted to the new workflow schema. In addition, for each workflow instance there may be multiple valid mappings, as described above. This default behavior of a workflow management system can be modified, for instance, if from an application-oriented point of view, workflow instances with a common characteristics should still use the original workflow schema. There are other options to deal with multiple workflow instances: If from an application point of view the adaptation shall only be done for a single workflow instance based on the original workflow schema, the other workflow instances can execute as if no new workflow schema existed. This flexibility is provided by retaining the original workflow schema, and executing the workflow instances normally.

7.4 Summary

This chapter investigates dynamic adaptations of running workflow instances to new workflow schemas. Dynamic adaptations are embedded in the context of the workflow application development methodology, as presented in Chapter 4. Two major approaches to workflow execution control are discussed: The build-time versus run-time approach, as found in most commercial workflow management systems today, and the interpretation-based approach. It it motivated that the latter approach is more adequate when it comes to dynamic adaptations, since the interleaving of workflow modeling (build time) and workflow execution (run time) as required for dynamic adaptations is supported by the interpretation-based approach.

It is not meaningful to allow arbitrary adaptations to running workflow instances, since the application process could be corrupted. Therefore, dynamic adaptations need to be controlled properly. While in general the correctness of workflow instances is ruled by the goals of the application process which is captured in workflow schemas, each workflow instance at any point in time is associated with one and only one workflow schema. This association, however, may change while the workflow runs. A correctness criteria which governs whether a workflow instance can be adapted to a new workflow schema is formalized. This criteria is based on the notion of valid mappings. In particular, if there is a valid mapping from the executed parts of the workflow instance under consideration to the new workflow schema then the workflow instance can be adapted to the new workflow schema. If such a mapping does not exist then the workflow instance can be continued with the original workflow schema.

Chapter 8

Implementation Issues

This section discusses the prototypical implementation of a workflow management system based on the conceptual design presented in the previous chapters. The design principles and the infrastructure are discussed in Section 8.1. Section 8.2 proposes the system architecture, while Section 8.3 introduces the design and use of fundamental services to implement the system. In particular, the design and implementation of a persistency service is explained in some detail. Section 8.4 shows how the design and implementation of the system relate to the design goals as presented in Section 6.1.

8.1 Design Principles and Infrastructure

Since the conceptual design of the system is based on object-oriented principles and covers the static and the dynamic aspects of the system in some depth, the system design rather closely resembles the conceptual design. Hence the workflow meta schema as specified in the class diagram in Figure 6.3 not only serves as a conceptual design to model workflows and related objects but also as an important basis for the design of a workflow management system. In fact, the classes shown in the workflow meta schema are implemented in the system. For instance, there is a generic workflow class, which holds workflow schema objects and workflow instance objects. The composition of complex workflows as a set of workflows with control flow and data flow constraints is specified by the WF-SubWF Relationship class and classes related to it, for instance the Control Connector class.

The close relationship between the conceptual design and the system design also applies to dynamic aspects. Since workflow instances are characterized by a complex dynamic behavior, each workflow instance object maintains information on its state and on its possible state transitions by implementing a state transition diagram. Using state tran-

sition diagrams, a workflow instance object responds to messages received depending on its current state. In particular, if a workflow instance object receives a message in a state for which that message is not defined (for example, a workflow object in the init state receives a message to be suspended) then the system raises an exception. Using specific code to handle these kinds of exceptions increases the robustness of the implementation.

While the general structure of the implemented state transition diagrams is given by the generic state transition diagram shown in Figure 6.8, the specific properties of automatic and manual workflow instances are represented by specific sub-states in state transition diagrams. For instance, transactional and non-transactional properties of external applications used to implement automatic atomic workflow instances are reflected in specific sub-states of the running state, called transactional and non-transactional. For detailed representations of these state transition diagrams, the reader is referred to [63].

In the process of developing a complex software system, the selection of an adequate computational infrastructure is an important topic, which may even effect the overall success of the implementation effort. Our decision to use an object-oriented approach for the conceptual design and the observation that workflow applications typically are deployed in distributed and heterogeneous organizational and technical environments naturally lead to the use of a distributed object middleware. Since standardization is an important prerequisite to interoperable software and, moreover, an important task of a workflow management system is the integration of external applications, we chose the CORBA middleware standard as an implementation platform (CORBA is the acronym for Common Object Request Broker Architecture). Using the CORBA approach, workflow schemas and workflow instances are both represented by CORBA objects, which communicate via messages. As a result, workflow execution control can be performed in a distributed fashion, and a dedicated centralized workflow engine is not required. This feature will be discussed in more detail in the context of distributed workflow executions.

While a thorough discussion of the concepts and technology of CORBA is outside the scope of this thesis — an excellent overview on distributed object middleware including CORBA is presented in [66] —, some important features of CORBA which affected our choice are briefly discussed. One important aspect is to re-use standardized fundamental services to implement the system, thereby reducing implementation and maintenance overhead of the software development project. These fundamental services are known as CORBA Services. In terms of platform independence and support for heterogeneous environments, CORBA provides the following features:

- Functionality is generally provided by objects: Client objects request services and server objects provide the desired services to client objects.

- Client objects and server objects can reside in different machines connected by a computer network running a CORBA environment.
- Client objects and server objects involved may be implemented in different programming languages.
- CORBA objects communicate via a software layer, called object request broker. An object request broker is a piece of software which provides a high degree of transparency to CORBA objects, including network transparency and location transparency.

Transparency is provided by the object request broker, using the CORBA Interface Definition Language (or CORBA IDL) in which the interfaces of CORBA objects are specified. We remark that each CORBA object resides in one site, and it has to be processed in that site. However in the workflow context, it can be useful to migrate objects between machines of different platforms. For instance, a workflow object may be created on a UNIX workstation to be processed on a PC system at a workflow participant's desk. Since these systems use different hardware and software, CORBA *a priori* does not provide support to execute methods of a given object on both kinds of machines. With the success of the Java programming language in recent years, it is possible to develop applications which run on a variety of systems in different hard- and software environments by interpreting a platform independent code, known as Java byte code.

Combining Java and CORBA, i.e., using Java to implement CORBA objects, is an attractive choice, since it allows to migrate objects between computers, e.g., from Unix workstations to PC systems and vice versa. Hence, implementing CORBA objects in Java increases the support for heterogeneity and mobility by the ability to execute CORBA objects in machines under different hardware and software architectures, allowing the migration of arbitrary CORBA objects as required by the application.

To summarize, based on an object-oriented design, the distributed object middleware CORBA and the Java programming language is an adequate choice to build a workflow management system, for the following reasons:

- *Object-Oriented Design:* Developing a workflow management system strictly based on object-oriented design principles is appropriate since the object paradigm provides powerful abstraction concepts which can be used in modeling the highly complex workflow domain.
- *Standardized Distributed Object Management Infrastructure:* Workflows are inherently executed in distributed and heterogeneous software and hardware environ-

ments; today, CORBA is the industry standard for distributed object management in heterogeneous environments.

- *Interoperability and Platform Independence:* By using a common infrastructure for OMG Business Objects [21, 26] and workflow objects, the integration of business objects into complex workflow applications is simplified. In addition, graphical user interfaces which allow users to access the system (workflow clients) can run on a variety of platforms, which is important since today's organizations typically use different platforms in their daily business.

The system is entirely written in Java, and most objects are CORBA objects, including workflow objects and related objects. While more than a dozen CORBA Services have been specified by the Object Management Group [80], few of them are commercially available today. Therefore, we have implemented a set of CORBA Services, which are required for the development project. While some of the services strictly implement the official CORBA specification as published by the Object Management Group, the interface specification of some services are altered, to avoid not needed complexity or to be able to implement the services more efficiently. The CORBA services, the core classes of the workflow management system, and the graphical user interface are implemented in Java. As indicated above, this approach allows to migrate workflow objects and the graphical user interface to different sites, according to the particular needs of the workflow application.

8.2 System Architecture

A high-level overview of the system architecture is shown in Figure 8.1. It is a layered architecture, composed of three levels: The applications level, the facilities level, and the foundation level.

The application level is the top level of the architecture. In this level, a workflow client application resides, through which workflow participants and workflow administrators access the system. In Figure 8.1, it is labeled by GUI, representing the graphical user interface of the workflow management system. In addition to the workflow client, external applications are also in this level, although they are not considered part of the system. Examples for external application programs are office applications like text processing or spreadsheet applications. These are examples of external applications which are invoked by the workflow client at the workflow participant's desk to implement manual atomic workflows.

Besides office applications, already existing information systems applications of the organization can be integrated. These information systems are often legacy applications, which have been implemented without workflow in mind. In this case, wrapper techniques have to be developed in order to integrate these applications in workflow applications. In this context, business objects are introduced, to allow for a seamless integration of external applications in workflow applications. While manual atomic workflows are performed by workflow participants using external applications, automatic atomic workflows do not require manual interaction of workflow participants. A typical example of an automatic atomic workflow is the execution of a program which accesses a database. Typically the parameters are fed to the program using data flow. Since no workflow participant is involved in this case, external application programs representing automatic workflows are invoked by workflow objects, rather than by the workflow client graphical user interface.

The facilities level is the core level of the system architecture. In this level, workflow objects and business objects reside. Besides workflow objects as described by the workflow class in the workflow meta schema, classes for the other classes are present. For instance, a WF-SubWF Relationship class holds objects which represent the structure of complex workflows, classes for horizontal and vertical data flow represent the relationships between parameter objects, as described in the workflow meta schema. In addition, classes for agents and roles hold information on the organizational context of the workflow management system. Data type and parameter instance classes keep track of the information aspect. In this level, the functionality of workflow schemas and workflow instances are implemented.

Workflow schemas can be created using the workflow client application in the upper layer. Once a workflow schema is created, it can be used to instantiate a workflow instance based on that schema. In this case, a new workflow instance object is created in the workflow class, with its accompanying parameters, control flow and data flow constraints, start conditions, etc. In addition, role information is defined by creating associations of a workflow instance object to a specific role object as specified in the workflow schema object. Once a workflow instance is started, these associations are used to select an agent (a workflow participant), to perform the workflow activity. To perform a workflow activity, a workflow participant is provided with a work item on the work item list of the workflow client application. Selecting that work item invokes the external application defined for that particular workflow activity. As will be shown below, there is additional functionality of the workflow client application, for instance to monitor running workflow instances and — of course — to dynamically adapt workflow instances to a new workflow schema. The implementation of the core functionality of workflow objects is described in some

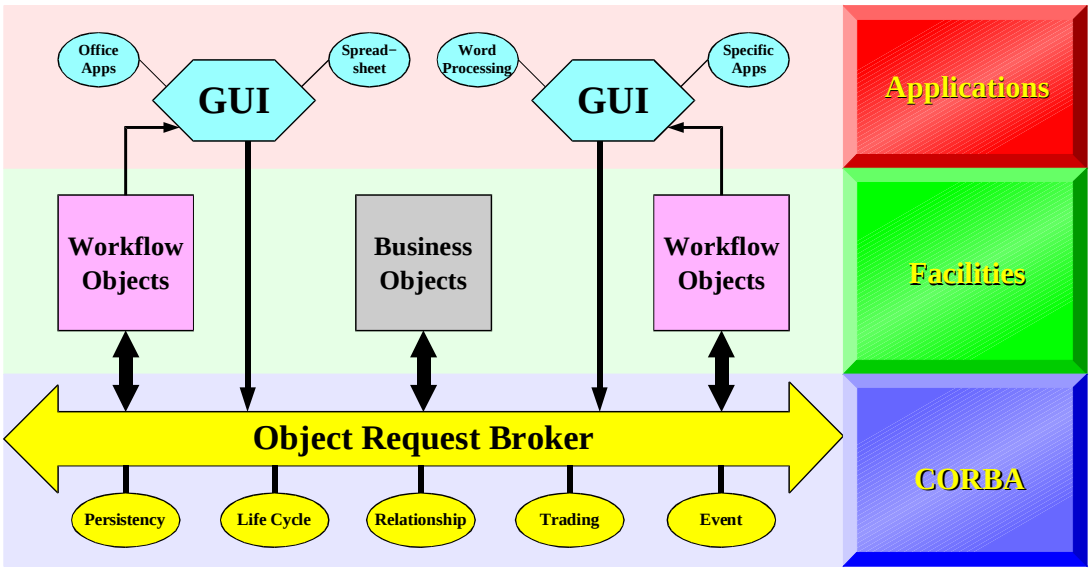

Figure 8.1: Architecture of Prototype.

detail in [63].

The basic objects and services to implement the higher-level objects are implemented in the foundation layer, shown in the lower part of the system architecture. As indicated above, this layer consists of CORBA Services. In particular, we have implemented a persistency service, to store arbitrarily structured workflow objects and related objects in stable storage [63]. A life-cycle service is responsible for managing the life cycle of objects, for instance to create, move, and delete CORBA objects [51]. Complex relationships between objects are maintained by a relationship service [92]. This service uses the role concept on the implementation level, so that each role a given object plays is represented by a role object maintained by the relationship service. For instance, a workflow schema object can play two roles: being the super-workflow schema of an atomic workflow schema and being the sub-workflow of a complex workflow schema. The trading service is responsible for locating objects in distributed settings according to specific properties of these objects [97]. Finally, the event service is used to collect and publish events which occur during the execution of workflows in distributed settings [40]. Using the concept of event channels, producer objects of events can publish events, while consumer objects can listen to event channels to get informed about relevant events.

The layered architecture provides a high degree of flexibility in maintaining the workflow management system. For instance, changes in the implementation of CORBA Services in the lower level do not affect the objects in the higher levels. This is an inherent feature of the object-oriented approach in which the behavior of an object is described by its interface, while the details of its implementation are hidden from the users of that particular service. Applying this general concept in the context of this project proved useful since, for instance, the implementation of the foundation services can be optimized without changing workflow objects in the facilities layer or the workflow client application in the applications layer.

8.3 CORBA Services

This section describes the design, implementation, and usage of CORBA Services in the context of the project. Since we found the OMG persistency service too complex for our needs and cumbersome to use, we have specified a persistency service which adequately suits our needs [63]. Based on this specification, we have experimented with different implementations of this service, based on both file systems and databases. The design and sample implementations of this particular service are discussed below. The lifecycle service [51], the trading service [97], the relationship service [92], and the event service [40], however, are based on the respective OMG specifications; therefore, no additional descriptions are required for the design and implementation of these services.

8.3.1 Usage in a Workflow Management System

The foundation layer of the prototype's software architecture consist of a set of CORBA Services. Their usage in the workflow management system prototype is described as follows:

The persistency service is responsible for storing arbitrarily structured objects in stable storage. This service provides atomicity properties for accessing objects in persistent storage. The aim of workflow management is the support for mission-critical business processes of an organization. Therefore, high availability and fault tolerance requirements are present in workflow applications. To cater for these requirements, workflow objects are generally persistent. This means that after a system crash occurs and object states in volatile storage are lost, the workflow objects can be restored to their correct values when the server is up again. This property is similar to recovery issues in database systems, where this functionality is common place today. Storing workflow schema objects and workflow instance objects in persistent storage is a requirement for restoring workflow

objects properly after a system restart. Durable objects have an additional application in the workflow context. The complete information on workflow instance objects is available after the workflow instance terminates. This property allows to analyze workflow instances which are completed. This is an important prerequisite for the analysis of workflow instances, i.e., for continuous process improvement which is an important activity in business process re-engineering. Information on performed workflow instances can also be used for quality control, since for each workflow instance the workflow participants, the external application programs, and the data objects involved are available.

The life-cycle service is used to create, copy, move, and delete CORBA objects. In the project, this service is used to manage the life-cycle of workflow objects and related objects. In particular, a workflow schema object is created when a new workflow schema is defined during workflow modeling [51]. A workflow instance object is created whenever a workflow is instantiated, typically followed by an external event, for instance the receipt of a customer order. Moving workflow objects between different sites of a CORBA environment is another function of the life-cycle service; this functionality is required whenever a workflow object has to be transferred to the location of a workflow execution, for instance to a workflow participant's desk. Without going into the technical details of its implementation, there are special objects, known as smart proxy objects, which are used to re-route a request to an object which has changed its location. From a system development point of view, moving objects between different sites of a CORBA environment is performed in a transparent way.

As discussed in some detail, there are complex relationships between workflow schema objects, workflow instance objects, and related objects, for instance role objects and agent objects. Whenever a workflow instance is created there is a SchemaOf-relationship between the workflow instance object and the respective workflow schema object; this relationship is maintained by functionalities provided by the relationship service [92]. This service allows to define role objects, which explicitly represent different roles a workflow object can play. For example, a sub-workflow is related to its super-workflow by the role *is_sub-workflow_of*. The same sub-workflow may also be related to its input parameter by the role *has_input-parameter*. Roles are a powerful mechanism to manage complex relationships between objects, and roles are a key element of the relationship service [80].

In complex CORBA-based applications, objects with specific properties are distributed among different locations. To support flexible queries to locate objects with specific properties, the trading service is used. For instance, locating agents with specific properties in distributed and complex organizations can be performed by the trading service [97]. This functionality is also used for flexible role resolution, which is based on the properties

and availability of agents. In particular, agents access the system by logging in, using the workflow client application. The system is informed of the agents which are logged in at any point in time, including the respective machines. The trader functionality is important in this context, since our approach is fully distributed, which implies the lack of a centralized site that keeps track of the system state, for instance of the availability of agents capable to perform workflow activities.

The event service implements complex event notification mechanisms between CORBA objects [40]. Based on the concept of an event channel, push communication and pull communication are specified. In push communication, for each type of event there are a number of producer objects (producers) and a number of consumer objects (consumers). Technically speaking, the producers put events into an event channel, and the consumers take the events from that channel. In the push model, consumers are actively notified whenever an event occurs for which they have subscribed. In the pull model, consumers retrieve events from event channels. In the project, push communication is used, e.g., in workflow monitoring: Each workflow instance sends relevant events to an event channel, for instance the start of a sub-workflow instance or its termination. The workflow monitoring tool is a consumer of these events. When the start event of a sub-workflow instance occurs, that event is inserted into the respective event channel. Since the workflow monitoring tool has subscribed to that event, the event is pushed to that tool. On receiving that event, the workflow monitoring tool is notified about the state change of the workflow instance, which is then displayed using the graphical user interface.

8.3.2 Design of Persistency Service

Persistent storage of arbitrarily structured objects is an important requirement for software systems. The design goals of the persistency service are specified by:

- *Ease of use*: To store a persistent object, that object inherits from the PersistentObject class. Then, by invoking a single method, the object state, i.e., the values of its attributes, is stored in persistent storage. If the attributes of the object are of standard data types, no additional coding is required. If the object contains complex values, the respective methods have to be overloaded.

- *Different data repositories*: By using a generic interface definition for the persistency service, different data repositories can be used to store objects persistently. For instance, relational database systems, object-oriented database systems as well as POSIX file systems can be used to implement the persistency service.

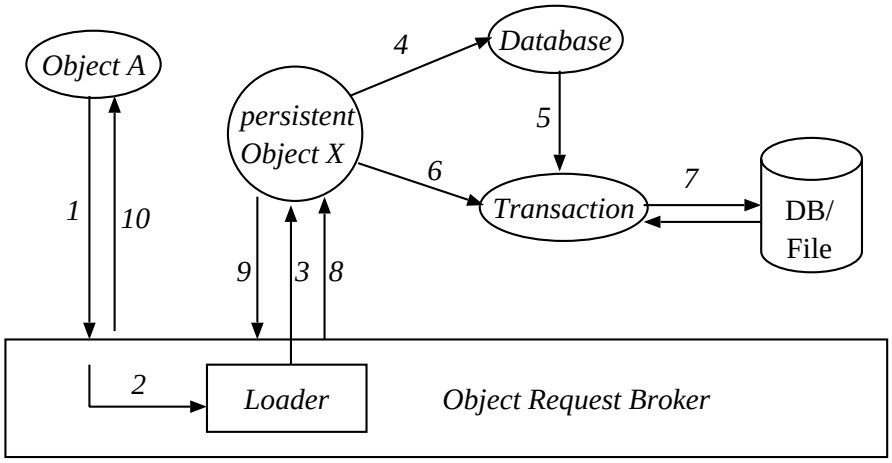

Figure 8.2: Usage of Persistency Service.

- *Distributed and heterogeneous data repositories*: The objects of a given CORBA application can be stored in different data repositories, which can even run on different machines, thus implementing distributed data repositories. In addition, different types of data repositories can be used to store objects of a given application. For instance, while some objects of a given CORBA application are stored in a file system, others are stored in a relational database.

- *Different object representations*: The interface definition allows different object representations in the data repositories. For instance, flattening objects, defining a single database relation per class, or translating an object into a binary stream and storing it in a database using binary large object structures are valid options to store objects in persistent storage.

After the general features of the persistency service are introduced, its functionality is explained by the steps which are performed to satisfy a request to a persistent object [63]; the resulting activities are shown in Figure 8.2. The request is triggered by an object A calling a method of a persistent object X, marked by (1) in Figure 8.2. If the requested object is available in volatile storage, the object request broker transfers the request to X. The request is processed, and the return values are transferred to the calling object A. If, however, persistent object X is not in volatile storage (2), the current object state of X has to be loaded from stable storage. Loading object states from persistent storage is the responsibility of the persistency service.

In this case, the object request broker re-initiates object X (3). At this point in time, this object is in the initial state, since the current values of its attributes have to be retrieved from persistent storage. The object request broker uses a loader component provided

by the implementation to instantiate that object. The loader determines the structure of the persistent object by accessing a database object, which returns the structure of the requested object.

Object X is responsible for restoring its actual state. To do so, X requests a transaction from the database object which is used to store X (4). The database object is a representation of the data repository responsible for storing X. In general, for each data repository there is one database object in the CORBA environment representing that data repository. Once the database object receives the call from persistent object X, it creates a transaction object (5), which is used to retrieve the data from persistent storage. In order to do so, object X sends a request data message to the transaction object (6), which accesses the data repository to retrieve the attribute values of the requested object from the data repository (7). In the next step, the actual values are restored in X, the object request broker sends the original method request to (the now restored object) X (8), which performs the method and returns the values (9) via the object request broker to the calling object (10).

After the operation of the persistency service is explained, some details on the design of the service are appropriate. The interface of a persistent object is described by the PersistentObject interface. Each persistent object has a dedicated database, which is its default storage location. The saveMe() method is used to store objects in persistent storage. To do so, saveMe() creates a transaction object and analyzes its structure using java.lang.reflect. The functionality provided by that standard software package allows any object to analyze its own structure. Technically speaking, a list of (attribute name, attribute type) pairs is returned, which describe the structure of the object's attributes with respect to Java standard data types. Complex data types can be analyzed by overloading the saveMe() method for particular classes. A boolean input parameter of saveMe() determines if failed operations have to be undone. Notice that schema adaptations involving just standard data types are transparent for the saveMe() method, since the structure of the object is retrieved during runtime using java.lang.reflect. However, if complex attributes are being changed, then the saveMe() method of the respective class has to be changed accordingly. An excerpt of the interface PersistentObject within the module CosPersistency is given in Figure 8.3.
After saveMe() has created a transaction object and has gathered information on the object's structure, it calls saveMeT() for each object to save. In our implementation, saveMeT() is performed for one object only. However, the interface definition allows to store other objects as well, e.g., to implement deep copies of objects, i.e., to store persistently the object that calls saveMe() and all objects which are referenced by that object recursively. In this case, all save operations will be performed in a single ACID transac-

```
module CosPersistency {
 interface PersistentObject {
  void saveMe(in boolean undo_when_failed) raises(Failed);
  void saveMeT(in Transaction t) raises(Failed);

  void loadMe() raises(Failed);
  void loadMeT(in Transaction t) raises(Failed);

  UnboundObjectRef getUnboundObjectRef();
  Database         getDatabase();
 }
};
```

Figure 8.3: Excerpt of Interface PersistentObject.

tion. On a technical level, saveMeT() calls a write-method for each attribute of the object to save; the write-methods are implemented within transaction objects, discussed below.

Analogous operations to saveMe() and saveMeT() are loadMe() and loadMeT(). The loadMe() operation creates a transaction object and requests its attribute values from that object. This is done by calling loadMeT(), which in turn generates a number of read-methods of the transaction object. These values are then transferred to the calling object, so that the persistent object receives the attribute values from the transaction object. On a technical level, saveMe(), saveMeT(), loadMe() and loadMeT() are Java synchronized methods. This means that within a single Java virtual machine, all synchronized methods are executed in a serialized fashion. In other words, these methods are performed in isolation of each other. Since furthermore the execution of each save operation is performed atomically, the execution of these methods are atomic. Hence, using the persistency service, reading and writing persistent objects is performed atomically. Transactional capabilities, however, have to be implemented by the objects or by a specific transaction service. When, for instance, an object representing the number of seats available in a plane is used in multiple applications then the object has to make sure that synchronization issues (like lost update situations) do not appear. Transactional issues are outside the scope of the persistency service.

The database interface describes the structure and behavior of data repositories, for instance file systems and database systems, i.e., it specifies how data repositories can be used to store persistent objects. The interface definition is generic in the sense that different database management systems as well as other data repositories can be used to store objects persistently. For each type of data repository, an implementation of the database

interface has to be provided. As indicated above, this feature allows to distribute the set of persistent objects of an application among multiple data repositories, which may even have different types. We have implementations for the Oracle relational database system and for POSIX file systems. An excerpt of the interface Database is shown in Figure 8.4.

```
module CosPersistency {
 interface Database {
  Transaction allocateTransaction() raises(Failed);
  void        freeTransaction(in Transaction t) raises(Failed);

  string   getNewObjectmarker() raises(Failed);
  Database findDatabase(in string marker) raises(Failed);

  PersistentObject bindMarker(in string marker) raises(Failed);
  PersistentObject bindUnboundObjectRef(in UnboundObjectRef obj)
   raises(Failed);
};
```

Figure 8.4: Excerpt of Interface Database.

When the persistency service is started, for each data repository, a database object is created, using the respective implementations. When a persistent object is accessed (i.e., stored or loaded), the object requests a transaction object from the database object, performed by the allocateTransaction() method. In this case, a transaction is opened by the database object, and the database object returns a transaction object to the persistent object, which uses it to process the request. The freeTransaction() method is used to terminate the transaction. When a new object has to be stored persistently, the getNewObjectmarker() method returns a string, which is then used as an object identifier for the new object. The bindMarker() method creates a CORBA object identifier structure, which means the object is registered in the object request broker. When accessing that object, it is loaded from persistent storage automatically, as described above.

The Transaction interface describes how transaction objects are structured. Transaction objects are used to store and load persistent objects. To do store an object, a write() method is run against the data repository. Hence, for each new type of data repository used, a system specific implementation of the Transaction interface has to be provided. Transaction objects convert language data types into data types of the data repositories used. As indicated above, the transaction interface supports the storing of multiple objects in a single transaction. This is done using a stack of objects to save. This feature can be used for recursively storing related objects. Whenever a link to another object is found,

that object is pushed onto the stack. Finally the stack is processed by popping object makers and storing the respective objects. In the current version of the persistency service, this feature is not implemented.

```
module CosPersistency {
 interface Transaction {
  void  pushWorkingObjectMarker(in string marker)
        raises(Failed);
  void  popWorkingObjectMarker();
  void  start() raises(Failed);
  void  commit() raises(Failed);
  void  abort() raises(Failed);

  void  writeObject(in string type, in string servername,
                    in string machinename) raises(Failed);
  void  writeAttrLong(in string name, in long value)
        raises(Failed);
  void  writeAttrFloat(in string name, in float value)
        raises(Failed);
  long  readAttrLong(in string name) raises(Failed);
  float readAttrFloat(in string name) raises(Failed);

  void  deleteObject() raises(Failed);
  void  deleteAttr(in string name, in string idltype)
        raises(Failed);
};
```

Figure 8.5: Interface Transaction.

To discuss the interface definition in more detail, storing an object starts with storing its meta information. This is done by the writeObject() method, which stores the server name (e.g., orbix_server) and the machine the server runs on (e.g., sunix). Storing objects is performed by storing their attribute values, which in turn is done by writeAttr<type>() methods. These methods are implemented for each standard data type of the language used, for instance, integer, long, string. These write methods are called by the saveMeT() method of the persistent object. The interface Transaction is displayed in Figure 8.5.

Transaction objects have commit() and abort() methods, which make sure reads and writes of objects are atomic. Depending on the implementation of transaction objects, these methods use the transactional feature of the underlying database. If a file system is used for persistent storage then the commit and abort functionality has to be provided separately. When a write to an object attribute fails then the writes of that object which already have been executed have to be undone, i.e., the original values have to be re-

installed. Failed reads are undone by re-installing the before image of the object in volatile storage. This functionality has to be supplied by the implementation of the transaction interface if non-transactional data repositories are used.

After introducing the functionality and the interfaces of the persistency service, different storage alternatives are discussed. Mapping object structures to non-object representations is an important issues in middleware technology. In general, there are different alternatives to store objects persistently, for instance, object flattening, one relation per class, and binary streaming. The interface definition of the persistency service allows to use any of these alternatives for each data repository used.

In the first alternative, for each attribute type of persistent objects, a database relation is created. For instance, there are database relations for long, float, and string data types. Each attribute of an object is stored in the respective relation, governed by its data type. This mechanism is called object flattening, since the structure of objects is not represented in its persistent storage representation. To store or retrieve a persistent object, all relations have to be accessed, which store attributes of that object. In particular, to store or retrieve a persistent object with attribute data types $t_1, t_2, \ldots, t_n$, relations storing values of these types will have to be accessed.

When it comes to system maintenance issues, this mechanism is very flexible, since new classes and new attributes can be added to the system without changing the underlying database structure. Adding new data types require only adding new database relations. Queries and updates can be performed using data manipulation languages (the Structured Query Language, SQL, for instance) and mechanism of the database system, which is convenient since it reduces implementation work, and the query optimization facilities of the database system can be used to access persistent objects efficiently. Finally, specific attributes of an object can be stored without the need to store the complete object. This feature reduces overhead when a complex persistent object is to be stored, but just small parts of that object are changed. In this situation, just the changed parts, i.e., a limited set of attributes of the object need to be stored. As a negative point, loading an object typically requires access to multiple relations, since the attributes of an objects are scattered among different relations, which may cause performance issues.

In the second storage alternative, all objects of a given class are stored in a specific database structure. In this alternative, for each attribute of a class, there is a column in a the relation representing that class. A mapping of the data types of the object level and of the database level has to be provided, just in the previous alternative. As a result, data manipulation languages like SQL can be used to access the attributes of objects. Accesses to the data are typically fast, since just a single relation has to be touched to access any

given object. However, the system is rather inflexible, since changing a class structure requires a change of the database structure.

The third alternative, the object information is encoded in a binary stream, known as binary streaming. This alternative is easy to implement, e.g., using the Java streaming mechanism. Access to data is quite fast, since the data of a given object is stored in one location. This approach is flexible, since object structures may change without changing the structure of the data repository. However, data manipulation languages like SQL cannot be used since there is no visible structure of the object in persistent storage, and objects always have to be accessed in their entirety. Therefore, implementing query mechanisms on top of this storage alternative may lead to poor performance.

Considering a class of thousand objects and a query that returns a few of them. Since there are no semantics of the data on the storage level, the thousand objects have to be accessed and loaded into main memory. To do that, the binary fields have to be analyzed to fill the attributes of the objects. Only in a second step, the query can be run on those objects, returning the desired result after complex computations. Assuming the query could use SQL, like in the second alternative, the query could be "pushed down" to the database system, in which case the database query processing system performs the query and returns the data for the objects in the result set. This mechanism is called query pushdown, and it used in the IBM San Francisco Framework [9] to perform queries. We remark that while these considerations are important for query processing in persistent object systems, it is not in the responsibility of the persistency service to provide query mechanism.

In the first implementation of the persistency service, runtime issues occurred due to poor performance of the underlying relational database system. This caused performance problems since all workflow objects and related objects are persistent and, consequently, use the persistency service. Then we decided to change the implementation of that service to use a file system. The design of the service was not changed. The respective classes of the persistency service were re-implemented within a short period of time, and the system worked with increased performance without any changes in the workflow classes. This of course is due to the object design and the separation of object interfaces and implementations. However, this practical experience demonstrated the usefulness of using object-oriented middleware in general and CORBA in particular. The specification of the modules for the two alternative implementations are given in Figure 8.6.

To conclude this section, some remarks are appropriate. The Object Management Group has specified a persistency service, which provides a high degree of flexibility in managing persistent objects. As a result, the design of this service is rather complicated,

```
#include "CosPersistency.idl"
#ifndef _DB_Oracle_idl
#define _DB_Oracle_idl

module DB_Oracle {
 interface Oracle_Database: CosPersistency::Database {
 };

 interface Oracle_Transaction: CosPersistency::Transaction {
 };
};
#endif

#include "CosPersistency.idl"
#ifndef _DB_File_idl
#define _DB_File_idl

module DB_File {
 interface File_Database: CosPersistency::Database {
 };

 interface File_Transaction: CosPersistency::Transaction {
 };
};
#endif
```

Figure 8.6: Interfaces for Storage Alternatives.

which leads to complex implementation and usage. To store an object of a given interface in persistent storage, a new DataObject interface has to be written, which takes care of reading and writing objects into stable storage. To create objects of a particular class, a DataObjectFactory class has to be specified and implemented for that class. To allow for a transparent creation of persistent objects, a GenericFactory has to be defined, which is aware of the DataObjectFactory of the objects to create. To summarize, the persistency service as specified by OMG is too complex for our needs, and it does not allow the usage of specific features of Java, which allows us to change the structure of objects without re-coding, as discussed above.

8.4 Design Goals Revisited

The design goals of our system are discussed in Section 6.1. We now elaborate on how the system design and its implementation relates to the design goals. This section is organized according to the design goals presented above.

8.4.1 Re-use of Workflow Schemas

Workflow schemas are represented by workflow schema objects of the workflow class, and each occurrence of a workflow schema as a sub-workflow schema is defined by a WF-SubWF Relationship object. Hence, each workflow schema can be re-used in an arbitrary number of complex workflow schemas. Since the embedding of these occurrences in the context of the complex workflow schema is defined using WF-SubWF Relationship objects (rather than workflow schema objects), each occurrence can have a different start condition as well as different control flow and data flow constraints. When a workflow schema is re-used, the system checks the resulting workflow schema for static correctness, i.e., it analyzes the structure of a complex workflow schema, the start conditions, and the data types of parameters, which are linked by data flow constraints. These properties are defined in the workflow meta schema on the conceptual level. The implementation as discussed earlier in this section also uses these concepts: each occurrence of a workflow schema object is represented by an object of the WF-SubWF Relationship class. Hence, the design of the system and the implementation provide adequate mechanisms for re-using workflow schemas.

8.4.2 Integration (and Business Objects)

As elaborated in Chapter 4, the integration of domain-specific application programs is a crucial success factor in workflow projects. An important step towards a seamless integration of these applications is defined by the notion of business objects. Business objects are coarse grained objects in the information systems which have a corresponding entity in the business domain [13]. Persons, bills, customers and warehouses are examples of business objects. Business objects are defined in [77] as follows:

> A business object is defined as a representation of a thing active in the business domain, including at least its business name and definition, attributes, behavior, relationships, rules, policies and constraints. A business object may represent, for example, a person, place, event, business process or concept. Typical examples of business objects are: employee, product, invoice and payment. The business object [...] is implemented by one or more objects in the information system.

Currently there are two approaches to standardizing business objects: While the Object Management Group is specifying a Business Object Facility [21, 26], IBM is proposing the San Francisco Framework [9, 10]. Both frameworks rely on a similar architecture, composed of layers of different functionality. Applications are on the top layer, which may use structure and functionality provided by objects which are likely to be present in most business applications, for instance in finance or human resources. These objects are called Common Business Objects. Common Business Objects in the OMG proposal use functionality provided by the Business Object Facility. Underlying functionality is implemented as the Foundation (IBM San Francisco), and as CORBA Services and other CORBA Facilities (OMG Business Object Facility), respectively. Concepts of business objects and business-object frameworks are discussed in more detail in [112].

The workflow management system is equipped to handle both kinds of business objects. From a technical point of view, business objects are manipulated during workflow executions. This allows to add workflow functionality to business objects. In particular, business objects can be transferred between workflows as data flow, and the functionality provided by business objects can be used to perform workflow activities. The usage of business objects in workflow applications can be represented in the workflow management system conveniently by defining that each input parameter and each output parameter of a workflow instance is able to hold a business object.

From a modeling point of view, business objects are in the Data Object class, as shown in the workflow meta schema in Figure 6.3. In particular, each business object class is

defined to be a sub-class of the Data Object class. Hence, each business object inherits from the Data Object class, which allows to treat each business object like any other data object used in a workflow application. The close integration of business objects in the prototype allows to use methods implemented in business objects to perform workflow activities. Hence, an activity performed during an atomic workflow can be determined by a business object which is transferred to that atomic workflow by data flow. This close link of business objects and workflow objects is important for a seamless integration of the application logic — as specified and implemented in business objects — and workflow objects that keep track of the application processes. Therefore, business objects are shown on the facilities level of the system architecture in Figure 8.1.

8.4.3 Dynamic Adaptations

Flexibility of workflow management systems is currently an important topic in workflow research and system development. While the exact range of flexibility issues are still under discussion (for instance, in recent workshops [61, 101]), it is accepted that dynamic adaptations, i.e., the ability of a workflow management system to adapt running workflow instances to workflow schemas, are an important aspect of flexibility in workflow systems. As explained above, the system supports dynamic adaptations of workflows by changing the SchemaOf-relationship between workflow instance objects and workflow schema objects. There are thorough consistency criteria which govern whether a dynamic adaptation can be performed, i.e., whether a workflow instance i can be adapted to a workflow schema s'. Dynamic adaptations are extensively discussed in Chapter 7, so that the design goal of dynamic adaptations can be regarded as fulfilled. We remark that dynamic changes have to be embedded in an organizational framework, which defines who has the competence to change workflow schemas and running workflow instances. The system, however, provides the mechanisms to support the required functionality on the technical level.

8.4.4 Distribution Aspects

There are two motivations for distributed workflow control. First, distributed workflow control matches well the organizational and technical environment in which workflow are inherently executed. Second, scalability issues are potentially less likely to occur in workflow management systems, in which workflow execution control is performed in a distributed way. These aspects are investigated in turn.

Today's enterprises operate in a distributed way, often on a nation-wide or even global

scale. This distributed organization has led to the development of distributed databases, in which the data of an organization is spread across multiple local databases, which in their entirety make up the global database. Each branch of the enterprise runs a local database system. This means that the branch is responsible for the operation of the system and the data stored in the system. Local autonomy refers to the high degree of self-determination which these branches have, and it is seen as one of the main advantages of distributed database technology.

These concepts also apply to workflow management. In a global enterprise, business processes involve multiple departments and branches, which may be geographically distributed. Just like in the database context, each branch is responsible for the parts of the business processes which are performed locally. Hence from an application point of view, it is meaningful that the local branches are also responsible for their processes. This involves their definition as well as their computerized support provided by a workflow management system. If local branches can perform their processes, or their parts of a global business process in an autonomous way, workflows are performed potentially more efficiently. In addition, dynamic adaptation of workflow instances, as described above, can only be performed using knowledge on the local business processes. Providing distributed workflow execution control puts the local branches of a global enterprise in the position to define, control and conduct their workflows. In addition, enterprise-wide workflows can be performed in a single system, which is distributed among the branches of the enterprise.

Besides these organizational reasons, scalability issues are also relevant in distributed workflow management. These considerations are based on the observation that large scale workflow applications today suffer from the fact that the workflow engine is often a centralized component, which is responsible for controlling the workflow instances in a given organization. This central component is likely to receive a heavy load if there are hundreds or thousands of workflow instances running concurrently, which is not atypical in large organizations. As a result, the workflow engine is likely to become a major performance bottleneck.

One effort to change this unsatisfying situation is partitioning of workflow engines. In this approach, the overall load of the workflow applications is partitioned onto distinct workflow engines, each of which runs on an individual server. In this scenario, however, new challenges emerge. In particular, the mapping of workflow instances to workflow engines far from obvious. In addition, overhead is introduced since workflow engines have to communicate with each other to provide workflow instances with information on the execution states of other workflow instances, which are controlled by a different workflow engine. Furthermore, workflow monitoring and process controlling typically

require information on multiple active workflow instances, which in general are dispersed among the set of workflow engines. In this situation, the monitoring information has to be gathered before it can be aggregated and displayed to the workflow administrator. These shortcomings are results of a centralized solution to a distributed problem, which workflow management is.

The workflow management system introduced in this thesis uses a distributed approach to workflow execution control, explained as follows: The execution of a complex workflow starts with the initiation of its immediate sub-workflows. Without going into the technical details, sub-workflow instances are instantiated as CORBA objects, and the control flow and data flow connectors between them are also represented by CORBA objects. The super-workflow sends a message to all sub-workflow instances which can be started. If vertical data flow is defined, the appropriate parameters are passed to the sub-workflows. The sub-workflow is executed and traverses states, as described by state transition diagrams. When the sub-workflow instance terminates, it sends a message to each workflow instance, which follows the terminated workflow in control flow. On receipt of this message, a workflow instance checks its start condition and begins execution, provided the start condition evaluates to true. This procedure continues until the last sub-workflow instance of the complex workflow terminates; this workflow sends a message to the super-workflow. On receipt of this message, the complex workflow is terminated. We mention that workflow objects can reside in different sites of a distributed computing system.

A workflow schema for which a distributed workflow instance is discussed below is shown in Figure 8.7. Top-level workflow schema 1 is composed of sub-workflow schemata 2, 3, 4, and 5, which are executed sequentially; the complete set of sub-workflow schemas and the control flow constraints between them can be seen in the figure. Not shown in that figure is the location of the individual workflow instances. There are different options for locating workflow instances to sites of a distributed system. For instance, a workflow instance representing an atomic workflow may be located at a computing site, which is close to the workflow participants who are expected to perform the respective activity. A complex workflow can be located at the server of the department which is responsible for performing the respective business process. Since the placement strategies are highly application-specific, they are not treated in depth in this thesis. The concepts and systems presented, however, provide the functionality for placing workflow instance objects on specific sites, as required by the application process.

In the prototypical implementation, the communication between workflow objects is done via the CORBA distributed object middleware. By distributing objects on different sites, the system scales nicely. The ORB, however, has to handle the communication re-

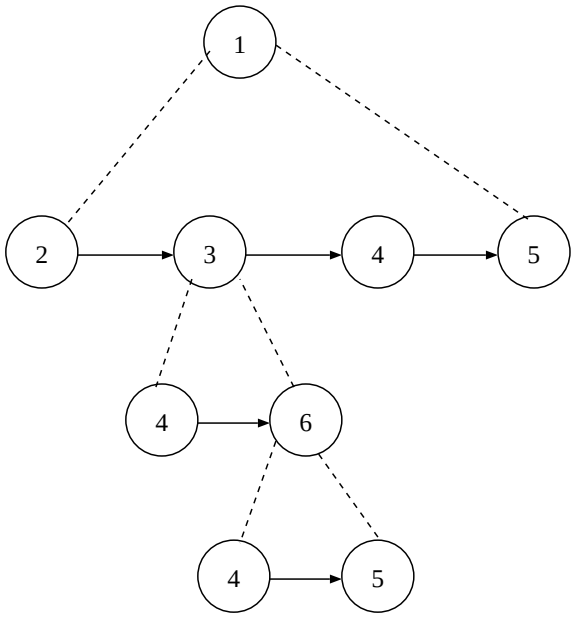

Figure 8.7: Workflow Schema, used for Distributed Workflow Execution Control.

quirements of the workflow objects. The Internet Inter ORB protocol (IIOP) allows the transparent communication of multiple object request brokers. This functionality can be used to implement company-wide and even business-to-business workflows. Scalability is provided by the distributed nature of workflow control and by the ability to migrate workflow instances to different sites of a distributed computing system to allow the balancing of processing load in large scale workflow applications.

A scenario of a distributed workflow execution based on that workflow schema is shown in Figure 8.8. That figure shows a complex workflow instance $1'$ composed of immediate sub-workflow instances $2'$, $3'$, $4'$, and $5'$; $3'$ is a complex workflow instance as is $6'$. The locations of the sub-workflow instances involved in top-level workflow instance $1'$ is shown by rectangles, representing sites 1, 2, and 3. In that figure, communication between workflow instances is represented by directed arcs. Workflow execution control is performed by workflow instances by sending and receiving messages. The top-level workflow is executed as follows: It starts by instantiating its immediate sub-workflows, as specified in the shallow instantiation approach. Then the first sub-workflow instance $2'$ to execute is sent a message, which evaluates its start condition and performs its work, implemented by invoking an external application. Notice that the communication is local in this case. After $2'$ terminates it signals the control flow connector, represented by sending a message to the next workflow instance, $3'$. Since this workflow instance is complex, at this point in time its sub-workflow instances are initiated. As shown in Figure 8.8, these

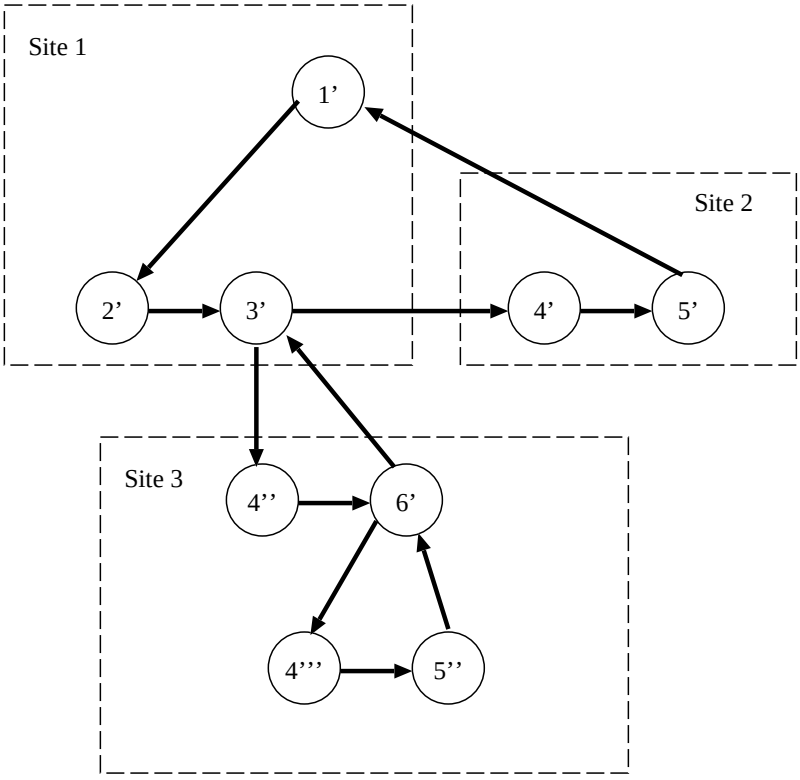

Figure 8.8: Distributed Workflow Execution Control.

sub-workflows are located in site 3. Hence, the respective workflow instance objects are created in that site. By that time, $3'$ sends a message to its sub-workflow $4''$ to start execution. This is a non-local message, from site 1 to site 3. Technically, message passing is done by the CORBA environment, so that the workflow instance objects do not require information on where the successor workflow instance objects are located. When that message arrives the workflow instance is started. The workflow continues as is specified in Figure 8.8, realizing distributed workflow execution control.

8.4.5 Persistency and Fault Tolerance

In the workflow management system introduced, workflow objects and related objects are persistent. More precisely, all objects whose classes are shown in the diagram representing the workflow meta schema (Figure 6.3) are persistent. As described above, the persistency service is used to store workflow objects in stable storage. Using persistent workflow objects is the prerequisite for fault tolerant workflow executions.

Based on the sample distributed workflow execution discussed above and shown in Figure 8.8, this aspect is exemplified as follows: Assume that after $3'$ sends a message

to its sub-workflow instance $4''$, residing in site 3, site 1 goes down, for instance after a power failure. The contents of the main memory of site 1 is lost. However, since workflow instances are persistent CORBA objects, the current state of the workflow instances is stored persistently. In particular, workflow instance $3'$ waits for its sub-workflows to complete, while $2'$ is already completed, and the top-level workflow instance waits for its sub-workflow instance to complete. Notice that currently no workflow instance is active in site 1. Since workflow execution control is performed in a distributed way, the other workflow instance can proceed while site 1 is down. In particular, the currently active workflow instance $4''$ can proceed in site 3, as if no site crash had occurred. When $4''$ terminates it sends a message to sub-workflow instance $6'$, which initiates its sub-workflow instances and starts executing by that time.

When $6'$ terminates, a message is sent to its super-workflow instance $3'$. Since this workflow instance resides in site 1, the object request broker forwards the invocation to that site. Assuming site 1 is up again, the object request broker process at that site recognizes that an object is requested (workflow instance object $3'$), which is currently not available in main memory. At that time the loader acts as described in the section on the implementation of the persistency service and as shown in Figure 8.2. Once the workflow object is loaded from persistent storage, the request is transferred to that object. The workflow continues with sending a message workflow instance object $4'$ at site 2. When $5'$ terminates, it sends a message to the top-level workflow instance $1'$, which terminates the workflow.

The procedure is similar when a site crashes which has active workflow instances. Since the state of workflow instances is maintained in persistent storage, workflow instances can be recovered after the system has restarted. If currently an external application program is executing to implement the workflow instance then the situation depends on the properties of that external application, in particular its transactional behavior. If the application is transactional in the sense that each execution of that application is performed atomically or not at all then the application will be aborted. When the system has recovered, the application is started again to implement the workflow activity. If the external application is non-transactional, then it has to be recovered manually. This situation is not different from crashing sites in non-workflow applications, where manual intervention is required to transfer the system into a consistent state after a system crash. However, since mission-critical applications are preferably implemented using transactional concepts, the automatic recovery of these important workflow activities is done automatically without human intervention.

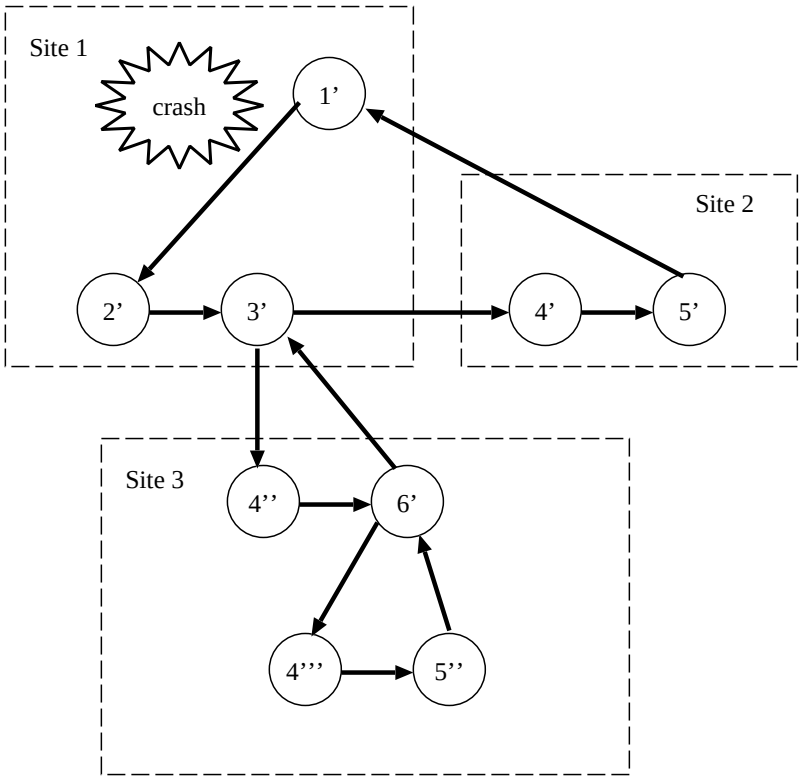

Figure 8.9: Distributed Workflow Execution Control and Crash of Site 1.

8.5 Summary

This chapter discusses the implementation of a novel workflow management system based conceptually on a formal foundation and an object-oriented design and technically on a distributed object-oriented middleware.

The architecture of the workflow management system is introduced, which is composed of a foundation layer, implemented by CORBA Services. The design and implementation of a persistency service is discussed in some detail. The main classes of the prototype are presented, and it is discussed how the design goals as formulated above are met by the design and implementation of the system. In particular, it is motivated that using a standard middleware technology is adequate since it allows distributed workflow execution control based on workflow objects which communicate to control their execution. In addition, the integration of domain-specific information systems can be improved by business objects. Business objects can be integrated well by defining data flow between workflow objects using business objects. Besides traditional data flow, by defining that an atomic workflow is performed by a specific method of a business object which is transferred by data flow, functionality provided by business objects can be used in workflow applications. The re-use of workflow schemas is supported since the occurrence of workflow schemas as sub-workflow schemas is represented by CORBA objects of the WF-SubWF Relationship class. The goal of dynamically adapting running workflow instances is satisfied by allowing a flexible assignment of workflow schemas and workflow instances and by consistency criteria which define if a workflow instance can be adapted to a workflow schema. Finally, distribution aspects are discussed, both with respect to the organizational motivation based on local autonomy of branches of an enterprise and with respect to scalability aspects. Persistency and fault-tolerant are supported by storing in persistent storage workflow instance objects, and by providing mechanisms to continue a complex workflow even if the machines of some of the sub-workflows involved have crashed. After restarting the machines, the correct values of the workflow instance objects are restored, and the execution of the complex workflow instance is resumed.

Some remarks on the technical environment of the implementation are appropriate. The system was developed in a cluster of SUN workstations running the Solaris 2.6 operating system. The object request broker OrbixWeb Version 2.0.1 by Iona was used. In that implementation, each machine of the CORBA environment runs a demon, called orbixd, which is responsible for the communication between CORBA objects. CORBA Services, workflow objects, and the workflow client application are developed in Java using Java Development Kit Version 1.1.7. OrbixWeb permits calls to CORBA objects from outside the CORBA environment. This functionality is used in the system by allow-

ing workflow client applications to reside outside of the CORBA environment. This is adequate from a technical and from a practical point of view: The machines of workflow participants accessing the system should not be part of the CORBA environment, which saves installation effort and, of course, license fees.

The system was presented to the general public at the Cebit computer fair in Hannover, Germany, in March 1999. For that demonstration, the CORBA environment was installed on a single SUN workstation, which also ran a workflow client application. A second workflow client application was installed on a PC, running MS Windows NT4.0. The system was also presented at the International Conference on Management of Data (SIGMOD) in Philadelphia, USA, in June 1999 [106]. Here the CORBA environment was installed on a laptop computer running MS Windows NT4.0 Server, which also ran a workflow client application, while a second laptop running MS Windows 95 hosted another workflow client application.

Chapter 9

Workflow Client Application and Sample Workflow

In this chapter, the usage of the workflow management system is shown in a sample application. After a sample workflow is discussed from an application point of view, the workflow client application is introduced as the front-end of the workflow management system. In particular, it is discussed how it can be configured toward different groups of users, ranging from workflow participants to workflow modelers and workflow administrators. The functionality of the workflow client application is illustrated using a sample workflow schema and a sample workflow instance. In particular, workflow modeling and the ability to monitor and to dynamically adapt running workflow instances are shown.

9.1 Sample Application Process

The sample application process is described as follows: An electronic bookstore runs a web site, which can be accessed by customers to select and order books. Customers access the web page with standard web browsers, and they select and order books using a forms-based interface. After a customer has filled in the order form, the information is checked for completeness. In case of incompleteness, the customer is asked to provide the missing information. If the information is complete, the order information is sent to the bookstore, where a workflow instance is initiated to represented the automated parts of the application process. The workflow instance is passed the order information data. Hence, the order of a customer automatically spawns a workflow instance in the electronic bookstore, and the order information provided by the customer is directly fed into that workflow instance.

As the workflow instance continues, it spawns a series of activities in the bookshop

to process that order. In the first activity, the order information as provided by the customer is checked for semantic validity. In the browser, the check was limited to syntactic completeness. If data is faulty, a double checking workflow instance is performed, which is used to correct the relevant fields in the order form. The next activity is a check of the warehouse of the bookshop. This activity is performed by running a query against the warehouse database. Assuming the ordered books are available, the parcel is shipped to the customer, and an invoice is prepared. The shipping of the parcel is performed by the shipping department of the bookshop, while the accounting department prepares and sends the invoice. While this description simplifies real-world application processes, it is well suited to present the workflow client application and the basic functionality provided by the system.

9.2 Workflow Client Application

Figure 9.1 shows the workflow client application, the graphical user interface of the workflow management system. The workflow client application provides different functionalities for different groups of persons involved in workflow modeling and execution. To support the requirements of these groups of users by a single application, the graphical user interface is configurable, allowing, for instance, workflow modelers, workflow administrators, workflow participants, and workflow supervisors to access the system with a single application. Depending on the role of the agent who logs on to the workflow client application, the different functionalities will be made available. For instance, a workflow participant will typically be presented a work item list, which is used to select work items, representing workflow activities to be performed. A workflow modeler requires a graphical tool to specify workflow schemas, while a workflow supervisor will need a workflow monitoring tool to keep track of the state of workflow instances. A workflow supervisor will require the ability to perform dynamic adaptations of workflow instances.

Hence, role information is not only used to select workflow participants to perform workflow activities, but also the configuration of the system is based on role information. This is a useful approach, since different workflow applications operate in different organizational environments and, hence, require different groups of persons to interact with the system. Role resolution with respect to the configurable workflow client application is performed as follows: When a user logs on to the system, the system accesses the role information for that user. Assume the user can play the role workflow modeler, the workflow client application provides the functionality to specify or modify workflow schemas. If a workflow supervisor logs on, a workflow monitoring tool will be present. Persons

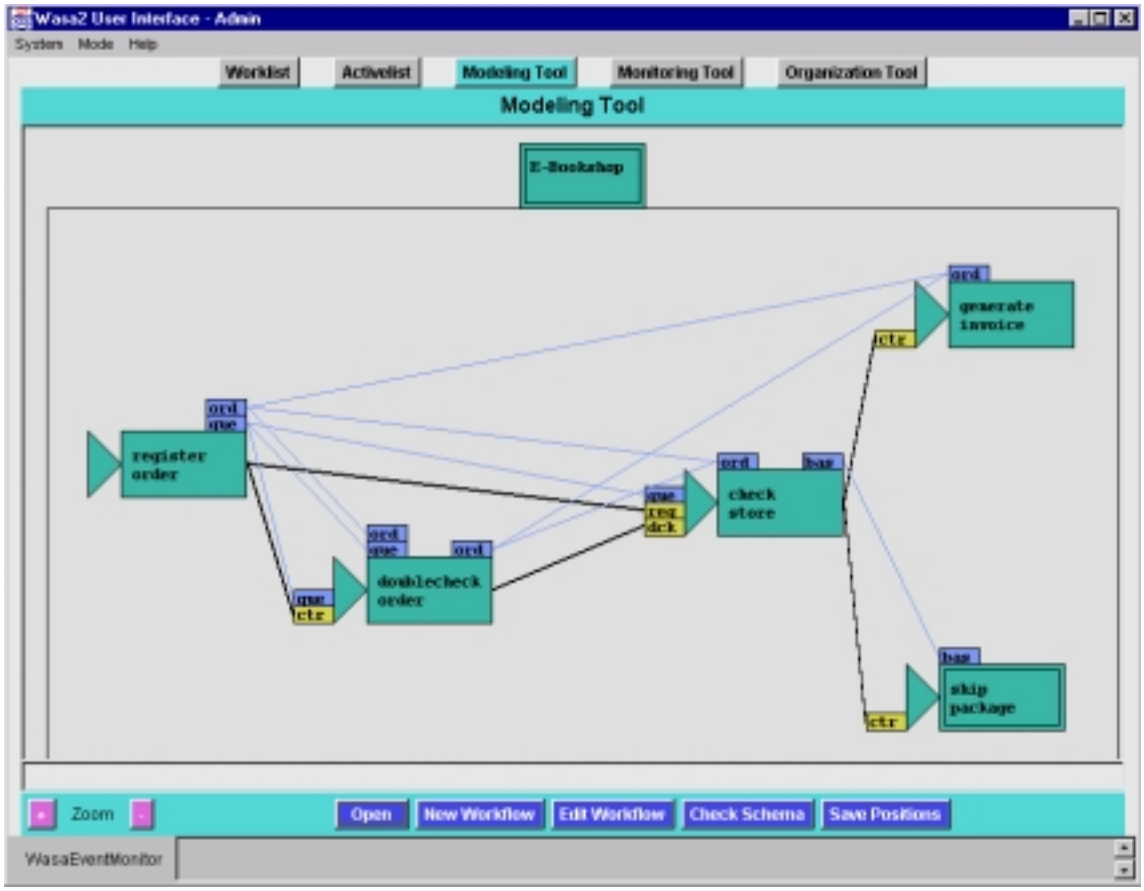

Figure 9.1: Electronic Bookshop Workflow Schema.

with multiple roles are presented the functionality of all the roles they can play.

In Figure 9.1, a list of buttons provide access to the functionalities for the groups of persons, as discussed above. In this case, a user with the role Admin (for workflow administrator) has logged on. Hence, a series of buttons are present in the workflow client application, which are listed in the upper part of that window. In this example, the workflow administrator can perform a variety of tasks, some of which may not be representative for workflow administrators in general. We opted for this choice to show the complete set of functions the workflow client application has to offer.

From left to right, a work list can be selected to show a list of work items, each of which represents a workflow activity which can be performed by the workflow participant (assuming the workflow administrator can also act as a workflow participant). An active list represents items for all workflow instances which a workflow participant currently works on or which a workflow supervisor currently controls. Notice that each user can concurrently work on or control multiple workflow instances, which may even belong to

different complex workflow instances. A workflow modeling tool allows the specification of workflow schemas, using existing sub-workflow schemas or defining new workflow schemas. Notice that Modeling Tool is highlighted in Figure 9.1, since that component of the workflow client application is currently active. The monitoring tool allows to monitor the progress of running workflow instances; the states of workflow instances — as defined in the state transition diagram in Figure 6.8 — are shown using a coloring scheme, as will be exemplified shortly.

An organizational tool allows the definition of agent and role information, to be used during role resolution. A history browser provides basic functionality to retrieve information on both currently running and completed workflow instances. It is possible to provide this information, since workflow objects are persistent in general, as discussed in the previous chapter. The history browser provides simple query mechanisms to access the historical information of workflow executions, e.g., by selecting all completed workflow instances based on a given workflow schema. This information can be used in process controlling to detect bottlenecks or other shortcomings in workflow schemas. We remark that the person who invoked the workflow client application shown in Figure 9.1 may not access the history browser; hence, no button for the history browser is shown. The design and implementation of the graphical user interface is discussed in [51].

To summarize, the following components are provided by the workflow client application:

- *Worklist:* The work list shows a list of work items, which a workflow participant can select to perform. Typically one work item represents a single workflow activity which is executed by a workflow participant using an external application program. If multiple workflow participants can perform a given workflow activity and one workflow participant selects the respective work item, the work item is purged from all other work lists.

- *ActiveList:* The active list shows all workflow activities of a user, which are currently active. Each workflow participant can concurrently work on a number of workflow activities, and a workflow supervisor can supervise multiple workflow instances at any point in time. The active list allows to keep track of the open workflow instances.

- *Modeling Tool:* The modeling tool is used for creating and modifying workflow schemas. A complex workflow schema can be created by re-using existing workflow schemas or by creating new workflow schemas. The canvas displays workflow schemas when the modeling tool is invoked.

- *Monitoring Tool*: Using the monitoring tool, workflow administrators can keep track of the progress of application processes represented by complex workflow instances.

- *History Browser:* The history browser allows to analyze active and already completed workflow instances. This component of the workflow client application makes use of persistent workflow instances.

- *Organizational Tool:* Using the organizational tool, workflow administrators can manage agent and role information, including application-specific roles of workflow participants and workflow-related roles, such as workflow supervisor or workflow administrator.

9.2.1 Workflow Modeling

The complex workflow schema representing the automated parts of the application process discussed above is shown in Figure 9.1. In that figure, the workflow modeling tool is selected, indicated by the highlighted button in the workflow client application. The canvas shows the electronic bookshop top-level workflow schema. More precisely, it shows the workflow schema node representing the top-level workflow schema and its immediate sub-workflows, i.e., it shows one level of the workflow schema graph as introduced in the mathematical formalization in Chapter 3. For layout reasons, workflow schema nodes are represented by rectangles, data flow is represented by blue lines, and control flow is represented by black lines.

Start conditions are shown as triangles attached to the left hand side of workflow schema nodes. This separation of workflow schema nodes and start conditions indicates that start conditions are not part of the sub-workflow schema, but start conditions are valid only in the context of a complex workflow schema, as explained in workflow schema reuse and as represented in the workflow meta schema in Figure 6.3. Start condition input parameters are shown as little boxes on the left hand side of start conditions. Input parameters and output parameters are represented by little boxes on the upper left and upper right, respectively, of workflow schema nodes. As is shown in Figure 9.1, control flow input parameters are yellow, and they are marked by ctr (control flow). If a start condition has multiple control flow input parameters then they are distinguished by their source workflow schema node. For instance, reg represents the control flow input parameter of the check store workflow schema, while dck represents the control flow input parameter of the double check workflow schema. It is no surprise that data flow connects output parameters of workflow schemas to input parameters of workflow schemas. Atomic workflow

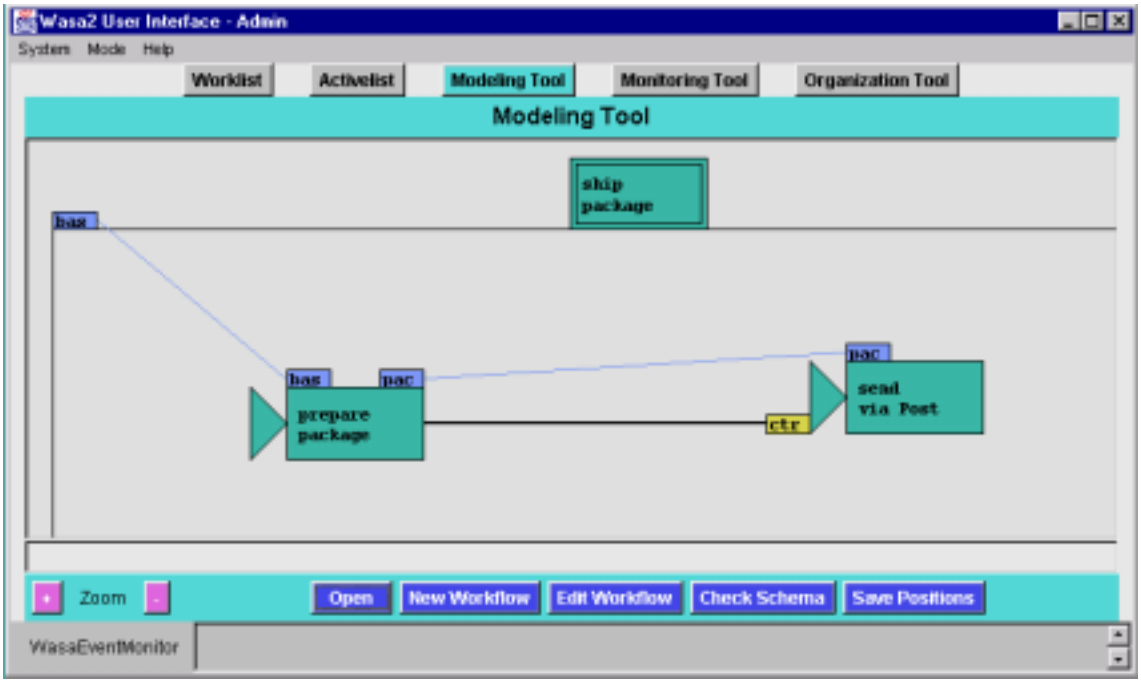

Figure 9.2: Workflow Schema: Ship Package.

schema nodes have single borders, while complex workflow schema nodes have double borders.

In the lower part of the workflow client application, there are zoom buttons for changing the size of workflow schemas or — as discussed shortly — of workflow instances. The other buttons are for opening existing workflow schemas (Open), for creating new workflow schemas (New Workflow) and for opening to edit an existing workflow schema (Edit Workflow). Selecting the Check Schema button invokes a schema checking algorithm, which checks the consistency properties of workflow schemas as specified in Definition 3.3.

After introducing the elements of workflow schemas in the workflow modeling tool, the sample electronic bookstore workflow schema is explained. Starting on the left hand side, the first sub-workflow schema is called register order. It receives the order information from the applet which is executed in the web browser of the customer who submitted the order. During this workflow, the order is registered, and a preliminary check is performed. The ord output parameter holds the order information, which is passed to follow-up workflows, as shown in the workflow schema. If something is wrong, a question is attached to the order, represented by the que output parameter. If a question is put, the double check workflow is performed, during which the order information is corrected. The check store workflow is an atomic automatic workflow. This means it is performed

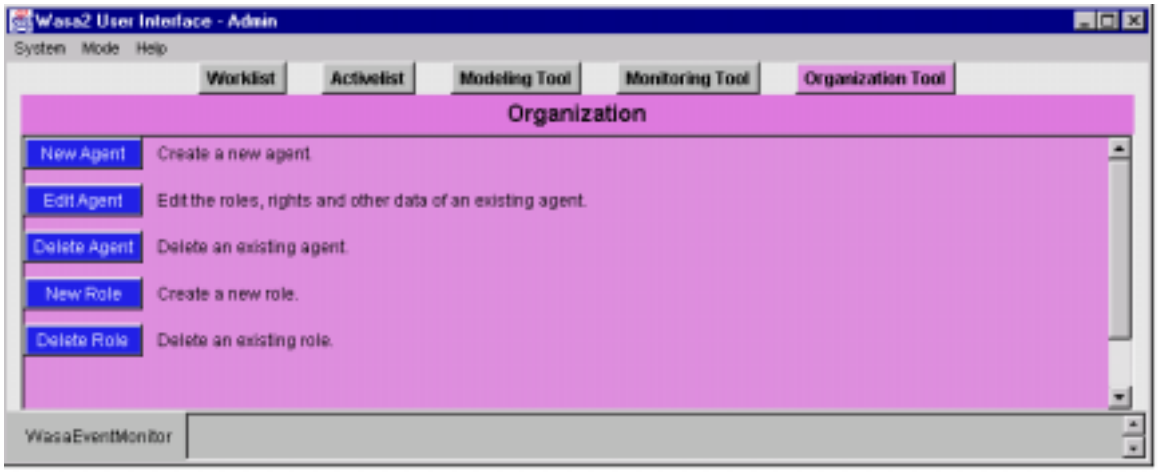

Figure 9.3: Organizational Modeling.

without human intervention by accessing the warehouse database. As a result, it generates an output parameter bas representing the basket containing the ordered books. The value of this structure is then passed to the ship package, which is responsible for shipping the package to the customer. Notice that there is no data flow from check store to generate invoice, since the pricing information is available before the check store workflow is performed.

The workflow schema ship package is shown in Figure 9.2; it consists of two sub-workflow schemas, with control flow and data flow constraints. In particular, there is a vertical control flow from the ship package workflow schema node to the prepare package sub-workflow schema. In this data flow, the information on the package to be shipped is transferred to the sub-workflow. Vertical data flows are represented by an edge leading from the input parameter of the complex workflow to the input parameter of the sub-workflow. In this case, a data flow edge links the input parameter of the ship package workflow schema to the input parameter of the prepare package sub-workflow schema. The second sub-workflow is send via post, which performs the actual shipping.

9.2.2 Organizational Modeling

While organizational modeling is not in the center of this thesis, the role concept is modeled in the conceptual design and is supported by the prototypical implementation of the system. The role concept provides an abstraction of the abilities, competences and organizational positions of persons involved in workflow applications. As indicated above, the organizational aspect is not restricted to the roles that workflow participants can play

during workflow executions. The organizational aspect also includes the fact that different groups of persons are involved in workflow applications. These organizational aspects are covered in the workflow management system described in this thesis by a configurable workflow client application so that the functionalities required for the different groups of persons can be adequately supported within a single workflow client application.

This approach has a number of advantages. First, it reduces overhead during implementation, installation and maintenance of the system, since the complete functionality is provided by a single application. Second, the functionality to configure the workflow client application allows to change the roles of persons during workflow applications easily. Consider a person, whose position changes from a workflow participant to a workflow supervisor. By re-configuring the workflow client application, the functionality to supervise workflow instances can be provided to that person. These changes can even be performed during the run time of the system. This ability of the system can be regarded as another facet of flexibility, since changes of workflow-relevant roles of agents can be conducted dynamically.

In general, organizational modeling is based on agents. Agents are persons involved in workflow-related activities in different stages of workflow projects, for instance workflow modeling, conducting workflow executions, monitoring workflow executions, and performing *ex-post* analysis of completed workflow instances. Figure 9.3 shows the workflow client application after the workflow administrator has logged on to the system and after she selected the organizational modeling tool. In the canvas shown, there are buttons for managing roles and the functionality required to configure the workflow client application for the different groups of persons involved in workflow applications. Using the workflow client application, new agents can be created, for instance after a new employee has been hired. Properties of existing agents can be changed using the edit agent button. Agents can be deleted if they no longer participate in workflow applications. A similar functionality is present for roles.

Figure 9.4 shows a window that allows to edit the properties of the agent Admin. A number of properties of that agent can be changed, for instance the password that the agent uses to log on to the system. The tools which are available to that agent can be selected from a list. In this case, all tools except for the history browser are available for the agent Admin. In the lower part of that window, role information of that agent can be edited. In this example, the agent can play roles accountant, clerk, shipping, and supervisor. Using that tool, roles for that particular agent can be added and deleted. The edit-role window allows to define a relationship of workflow schemas and roles. For example, the invoice is generated by an agent with role accountant.

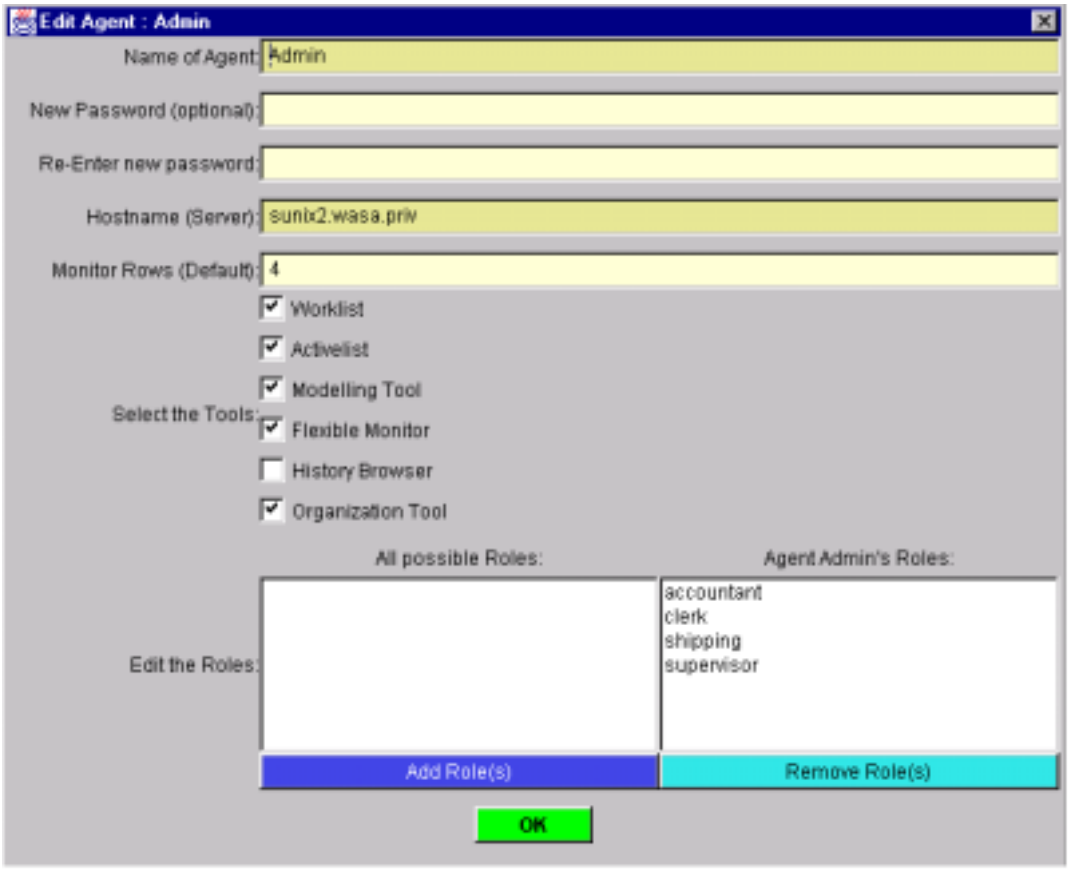

Figure 9.4: Configuration of Workflow Client Application.

9.2.3 Workflow Monitoring

A workflow instance based on the electronic bookstore workflow schema is shown in Figure 9.5, using the monitoring tool. Notice that the structure of the workflow instance is similar to the structure of the workflow schema. However, the states of the sub-workflow instances are represented by a coloring scheme. The state changes of workflow instances are visible by changing colors of the workflow objects. This allows to keep track of the progress of workflow instances while they execute.

Figure 9.5 shows the workflow instance after the register order and check store sub-workflow instances have already terminated, indicated by a blue coloring of the respective workflow instance nodes. The doublecheck sub-workflow instance was not performed during that particular top-level workflow instance, i.e., it was eliminated, which is represented by a dark blue coloring. The ship package sub-workflow instance is in the running state, which is indicated by the red coloring. The generate invoice sub-workflow instance is in state ready, which means that there is a work item for that sub-workflow instance on the work list of a workflow participant. The ready state is represented by a green coloring. When a workflow participant selects a work item representing that particular atomic workflow instance, the state change from ready to running is represented by a color change from green to red. By selecting workflow instance nodes using a pointing device, the current state of that workflow instance is specified textually in an information field of the workflow client application; this information is not shown in the screen shot.

Besides displaying the state of sub-workflow instances, the workflow monitoring tool allows to display values of data objects transferred by data flow. To do that, the user puts the pointing device in the box of the parameter which contains the value. If the value is atomic, it is displayed in the lower part of the window. Strings and numerical values can be retrieved directly using the workflow monitoring tool. If the value is complex then a single line will not suffice to display it. In this case, additional tools can be started to display its value. For instance, if a business object is transferred between workflow instances then a display() method implemented in that particular business object can be used to show its value. This functionality of the graphical user interface allows not only to keep track of the progress of workflow instances, but also to inspect values of data objects created or changed during workflow applications.

9.2.4 Dynamic Adaptations

After dynamic adaptations have been discussed in detail from a theoretical, conceptual, and implementation point of view, this section introduces the usage of a dynamic adaptation, based on the electronic bookshop sample workflow. Dynamic adaptations can occur

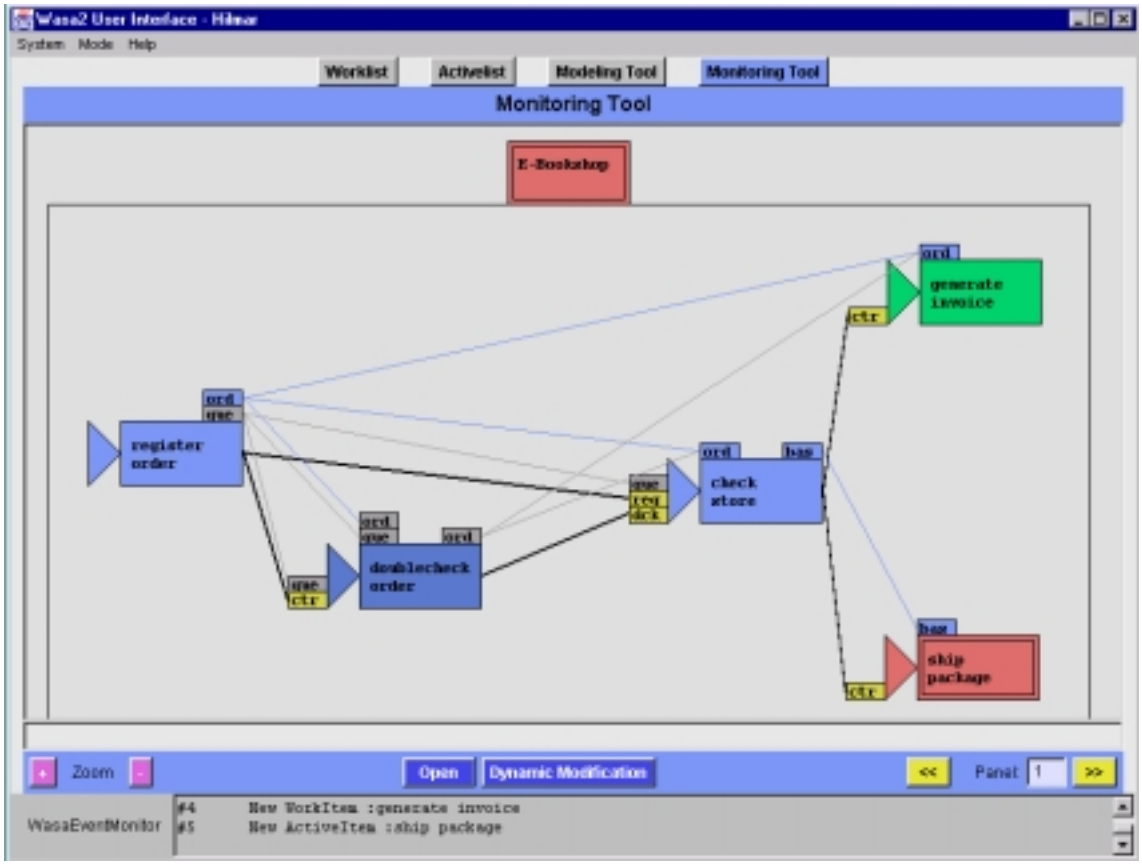

Figure 9.5: Electronic Bookshop Workflow Instance.

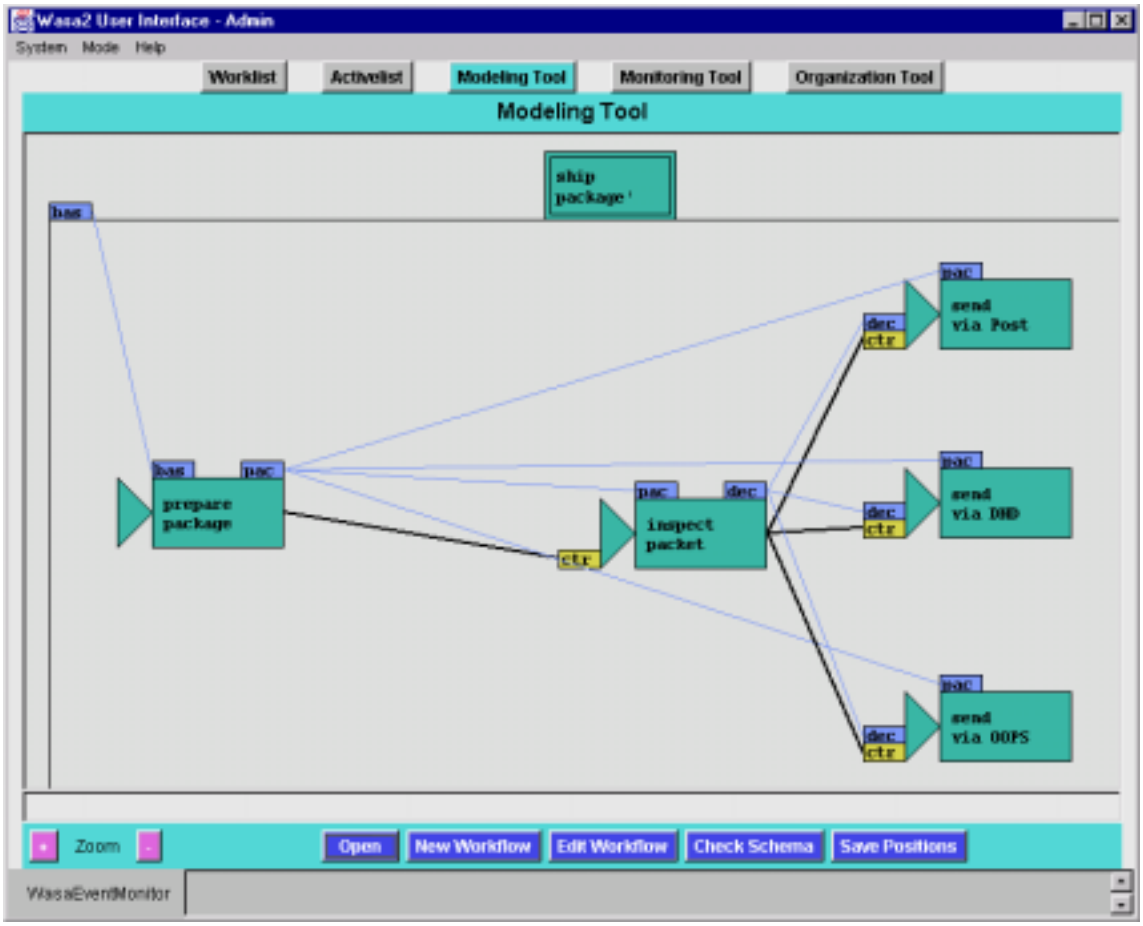

Figure 9.6: New Workflow Schema: Ship Package′.

in top-level workflows, but they can also appear in complex sub-workflows. This example considers a change of the latter form: The complex sub-workflow instance based on the ship package workflow schema is adapted to a new workflow schema. To introduce the dynamic adaptation example, reconsider the original workflow schema ship package, as shown in Figure 9.2.

Imagine a situation in which new competitors emerge in the postal service market, and the electronic bookstore wants to make use of these new services for the current workflow instances and for all future ones. The new workflow schema is shown in Figure 9.6. As shown, the new workflow schema differs considerably from the original one. First of all, an inspect package workflow is added which checks the parcel and determines which of a set of postal services to use for a particular workflow instance. Finally, there are three sub-workflow schemas, one for each postal service available.

To check if a dynamic adaptation is feasible, the system checks whether the workflow

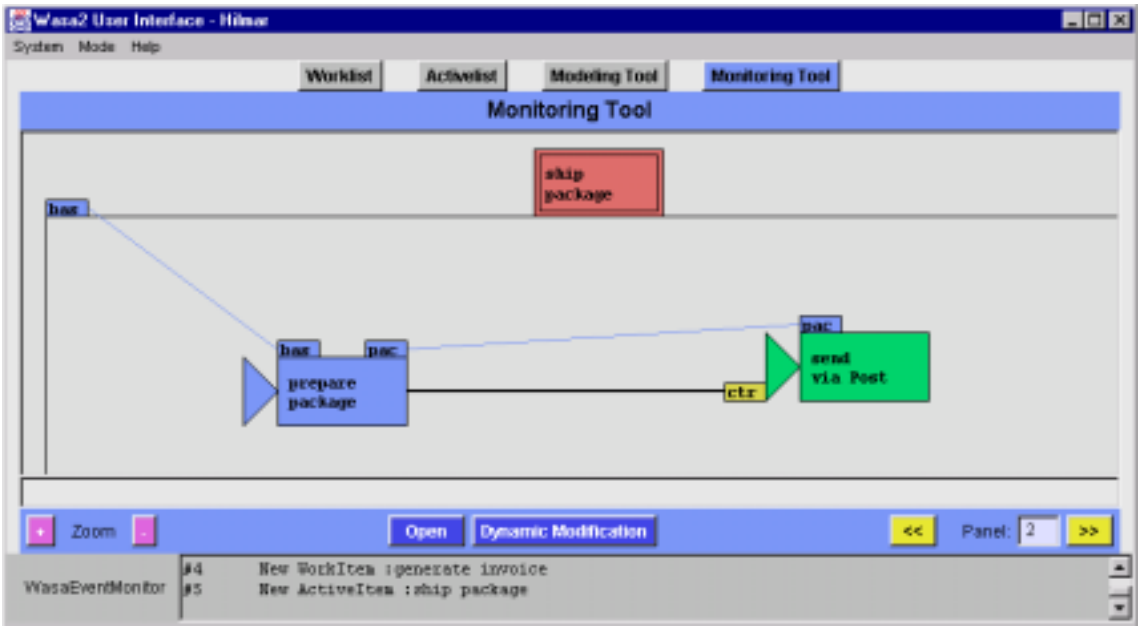

Figure 9.7: Workflow Instance to Adapt Dynamically.

instance under consideration can be continued to become a complete and correct workflow instance with respect to the new schema. As is shown in Figure 9.7, the workflow instance has already executed its first sub-workflow called prepare package. The system checks if the workflow can be continued using the new workflow schema. Intuitively this condition is satisfied, since the only sub-workflow instance completed is the prepare package workflow, which is also present in the new workflow schema, and all added sub-workflows will be executed after it. From a system's point of view, this correctness check is performed by computing valid mappings, as specified in Definition 7.4. The system finds a valid mapping, and the workflow administrator can now select it to perform the dynamic adaptation of the workflow instance.

Now that preliminary considerations for using dynamic adaptations are complete, one particular sub-workflow instance using the ship package workflow schema is chosen; this complex workflow instance is shown in Figure 9.7. As indicated by the coloring scheme, the first sub-workflow instance of that complex workflow instance is already completed, while the second one, the send via post workflow instance, is in the ready state, i.e., it is currently represented by a work item in the work item list of a workflow participant. Notice that this workflow instance has not started yet. Since the ship package sub-workflow has already started, the current workflow instance has to be changed dynamically in order to make use of the new postal services.

At the start of the dynamic adaptation, the send via post sub-workflow instance is

suspended. As a result, the respective work item is purged from all the work lists of all workflow participants it was assigned to. Now the workflow supervisor selects a workflow schema to which the workflow instance shall be adapted. Any workflow schema can be chosen here. However, the consistency check makes sure that a workflow instance i will be adapted to a workflow schema s', if i is adaptable to s'. In order to decide on the adaptability, the system computes a set of valid mappings and presents them to the workflow supervisor. Recall that there may be multiple valid mappings for a given dynamic adaptation. Assuming the workflow supervisor is responsible for the dynamic adaptation, he then selects one valid mapping as a basis for the dynamic adaptation. In this example, there is just one valid mapping: The completed sub-workflow instance prepare package is mapped to the workflow schema prepare package of the new workflow schema ship package′.

Notice that control flow and data flow properties defined for valid mappings are also satisfied in this example. Hence, the workflow instance can now be adapted to the new workflow schema ship package′ to make use of the new postal services. To implement the dynamic adaptation, workflow instances for the sub-workflows are initiated, including their parameter and their control flow and data flow constraints, as is discussed in [92]. Figure 9.8 shows the dynamically adapted workflow instance, as it continues.

9.3 Enhancements of Workflow Client Application

The facilities layer of the system architecture provides the objects and the functionality required for various aspects of workflow management, including creating workflow schemas, enacting and controlling the execution of workflow instances, and dynamically adapting workflow instances to new workflow schemas. This functionality is provided by the workflow client application. In this section, enhancements of the workflow client application are discussed, most of which can be implemented without changes in the facilities layer.

In the current version of the workflow client application, its use as an information tool is limited to monitoring individual workflow instances. Workflow monitoring is an important functionality since the supervisor of a complex workflow instance can monitor the progress of that workflow instance. However, besides workflow monitoring there are other kinds of activities which are relevant in this context. For instance, the simulation of workflows during workflow modeling. The goal of simulating in general is to be able to analyze the expected effects of different proposals to the real world without actually performing activities. In the workflow context, the logical structure of application processes

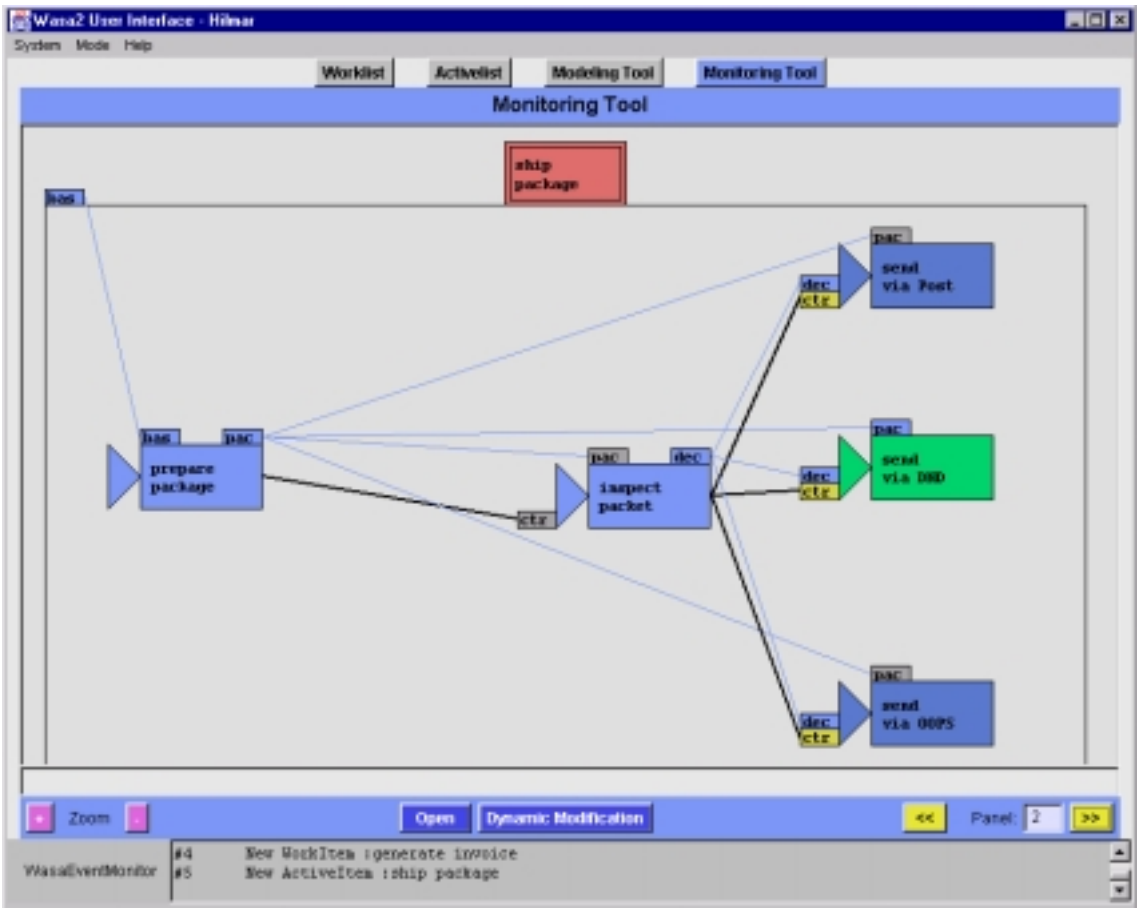

Figure 9.8: Dynamically Adapted Workflow Instance.

as specified by workflow schemas can be simulated with a simulation tool. In appearance, this tool is quite similar to a workflow monitoring tool, since it considers the workflow instance level. To validate a workflow schema, the workflow modeler will choose the simulate option from the workflow client application. In effect, the execution of a workflow instance is simulated. Useful features of the simulation tool are the ability to change workflow relevant data and application-specific data in order to simulate a wide range of workflow instances.

Simulation can even go one step beyond. By attaching expected duration times to workflow instances and the number of persons available to perform certain roles, quantitative simulation can be performed. In this case, probabilistic assumptions are taken for application data values. For instance, 60% of all credit requests have an amount not exceeding a certain threshold. The simulation tool will then simulate the execution of multiple workflow instances and display potential bottlenecks of workflow executions based on the workflow schema. These simulations need not be limited to a single workflow instance. On the contrary, a number of workflow instances of a given organization can be simulated. To support this functionality, quantitative figures have to be modeled in the workflow meta schema, for instance the expected duration times of workflow instances and the number of persons ready for playing certain roles.

A related functionality of a workflow client application is the analysis of workflow instances which have already been executed. The basic requirement for such an analysis, i.e., the persistent storage of workflow instance, is satisfied by the workflow management system. As indicated above, we have built an initial version of a tool called history browser, which allows to compute and visualize certain properties of all workflow instances of a given workflow schema. Useful functionality in this context is the analysis and visualization of the distribution of the execution durations of workflow instances. Also the performance of workflow participants can be visualized and compared to the performance of others. This functionality is possible from a technological perspective. However, as was indicated in the section on workflow application development processes, acceptance issues in workflow management have to be taken into account from a sociological point of view. The functionality in this context will have to be discussed with the groups involved in organizing the work in an enterprise, both from an employer's point of view and from an employee's point of view.

In the context of the organizational aspect, the system can be enhanced to provide more complex role resolution policies, for instance advanced substitution policies, delegation of responsibilities, and value-sensitive role resolution. These concepts are sketched as follows:

- *Substitution Policies*: Given a complex organizational model, describing the organizational structure of an enterprise (involving, e.g., branches, departments, groups, and projects), role resolution can be based on specific substitution policies. For example, the manager of project A will substitute the manager of project B if B is a sub-project of A. However, more complex rules on substitution policies can be described, depending on the specific policies of the application.

- *Delegation of Responsibilities*: There may be situations during the execution of workflows in which workflow participants selected by role resolution are not able to process workflow activities assigned to them. In this case, the respective work activities can be delegated to other workflow participants, who are more experienced with the particular kind of workflow activity requested.

- *Value-sensitive Role Resolution*: A popular example to motivate value-sensitive role resolution is the granting of a vacation request. This example was already sketched above. To grant a vacation of an employee A, the manager of A is responsible. Unless, of course, the vacation request was put forward by the manager herself. In this case, the manager of the requester has to decide on the vacation request. In this case, a value transferred between workflow activities by data flow, i.e., the name and position of the requester, has to be taken into account during role resolution.

While neither of the three sample concepts for advanced role resolution are implemented in the current version of the workflow client application, all of these could be provided with little effort. For example, the substitution policy requires the organizational structure of the enterprise to be represented in the system properly. Then, the substitution policies are specified as rules, and the role resolution methods have to be overloaded in order to implement these policies. To support the delegation of activities, additional functionality is required for the workflow client application, e.g., for transferring work items between the work lists of workflow participants. The delegation has to be restricted by rules on which particular workflow participant is allowed to delegate which workflow items to whom. Value-sensitive role resolution can be implemented by overloading the role resolution method. In addition, the functionality to provide rules (like the one described above) has to be implemented in the workflow client application. This discussion suggests some enhancements of the system to improve the functionality of the system with respect to organizational modeling; due to the open design and the layered architecture of the system, enhancements can be performed as required by specific workflow application development projects.

9.4 Summary

This chapter discusses the usage of the workflow management system based on a sample workflow. In particular, a workflow client application is presented, which is the graphical user interface to the workflow management system. Since there are multiple groups of persons involved in modeling, executing, and monitoring workflows, the workflow client application is configurable. The role concept is used for specifying which persons are presented which functionality of the workflow client tool. For instance, when a person logs on to the system, who is assigned the role workflow administrator then the complete set of functionalities of the workflow client application is present. If, however, a workflow participant logs on, just the work item list and the activity list are shown. By changing the role information, changes in the organizational structure of the company can be represented. Workflow schemas can be specified interactively using the workflow modeling tool. The progress of workflow instances can be controlled with a workflow monitoring tool. This tool uses a coloring scheme to visualize the states and state changes of workflow instance nodes during the execution of complex workflow instances, represented by workflow schema graphs. Dynamic adaptations are exemplified using a sample workflow instance. Correctness criteria formally defined above are implemented to make sure a dynamic adaptation of a workflow instance to a new workflow schema is conducted only if it is meaningful. This chapter concludes with remarks on enhancements of the workflow client application.

Part III

Related Developments and Future Work

This part describes related developments and future work. Since the topics presented in this thesis cover a wide range of workflow management research issues, related work is presented in categories. Workflow languages and flexibility aspects in workflow management are an important contribution of this thesis. Hence, related work on these aspects is discussed first. There are a number of related research projects in the areas of object-oriented and distributed workflow management systems, which will be discussed in Section 10.2. Section 10.3 sketches related work in the area of workflow management in science and engineering. Finally, related work on design methodologies for workflow applications is discussed.

Chapter 10

Related Work and Developments

This chapter discusses related work and developments in the areas of workflow languages and flexible workflow management, object-oriented and distributed approaches to workflow management systems development, and workflow management in science and engineering. Finally, design methodologies aiming at improving the planning and conduction of workflow projects are discussed.

10.1 Workflow Languages and Flexibility Aspects

This section discusses related workflow languages and the support for flexibility provided on the conceptual level or — if implementations exist — by the respective workflow management systems. In particular, a commercial workflow management system aiming at supporting production workflows is considered, and workflow programming languages are addressed. Approaches dedicated to flexibility issues are discussed in some detail; they include the ADEPT project, work on workflow evolution, and an approach based on a specific form of Petri nets, tailored towards the requirements of workflow management. Finally, a number of recent developments in flexible workflow management are sketched.

MQSeries Workflow

One of the most widely used commercial workflow management systems is IBM's MQSeries Workflow [53], formerly known as IBM FlowMark [53]. This section sketches the workflow meta model of MQSeries Workflow and the graphical notation used to specify workflow schemas, while its textual representation in Flow Definition Language (FDL) will be overviewed below. Like the approach to specifying workflows presented above, the MQSeries workflow language is based on nested directed graphs, called pro-

cess model graphs. In process model graphs workflows are represented by nodes, while edges represent constraints between workflow nodes. To describe the MQSeries workflow language, the main components of its workflow meta model are discussed [53, 67, 68].

The main entities in the MQSeries workflow meta model are activities, which can be complex or atomic. Complex activities are called processes, while atomic activities are called program activities to indicate that atomic workflows are typically implemented by external application programs invoked by the workflow management system. Processes are composed of a number of activities with accompanying control flow and data flow constraints. A set of activities can be grouped using the block construct. The activities in a block are executed repeatedly until a termination condition of the block signals its completion. More precisely, after each iteration of the block's activities, the termination condition of the block is evaluated. The block is re-iterated if and only if the termination condition evaluates to false.

With regard to the information aspect, each activity has an input container and an output container, consisting of a set of input parameters and output parameters, respectively. In MQSeries Workflow, data flow is specified by connecting an output parameter of an activity i to an input parameter of an activity j. This data flow is permitted only if there is a path of control flow edges from i to j. Control flow is defined by control flow edges. Each control flow edge is assigned a transition condition, which is a predicate to be evaluated at runtime. When an activity terminates, the transition conditions of all outgoing control flow edges of that activity are evaluated. Depending on the value, the control flow edge is fired with either true or false values. The start of activities is governed by start conditions. In general, start conditions of an activity can be evaluated if and only if all incoming control flow edges have been fired. When a start condition evaluates to true, the respective activity can be launched. To guarantee that control flow edges fired false do not hamper the start of workflow activities, a technique called *dead path elimination* is performed [67]. Workflow schemas in MQSeries Workflow are generally acyclic; iterations can be modeled using the block construct, as discussed above.

A sample workflow is specified graphically in the system as shown in Figure 10.1, using a screen shot. Activities are represented by nodes, data flow is represented by dotted lines, and solid lines are control flow edges. Control flow edges are marked with transition conditions, which are evaluated on termination of the source node of the respective control flow edge. That figure shows a characterization of the credit request workflow. To model that a credit request can be granted if less than 100 K$ is requested and if the risk factor determined by the AssessRisk activity is low, the transition condition from the AssessRisk to the AcceptCredit activity in Figure 10.1 is labeled "CreditAmount $<$ 100000 AND RiskFactor="L"". Hence, the behavior of workflow instances is described by tran-

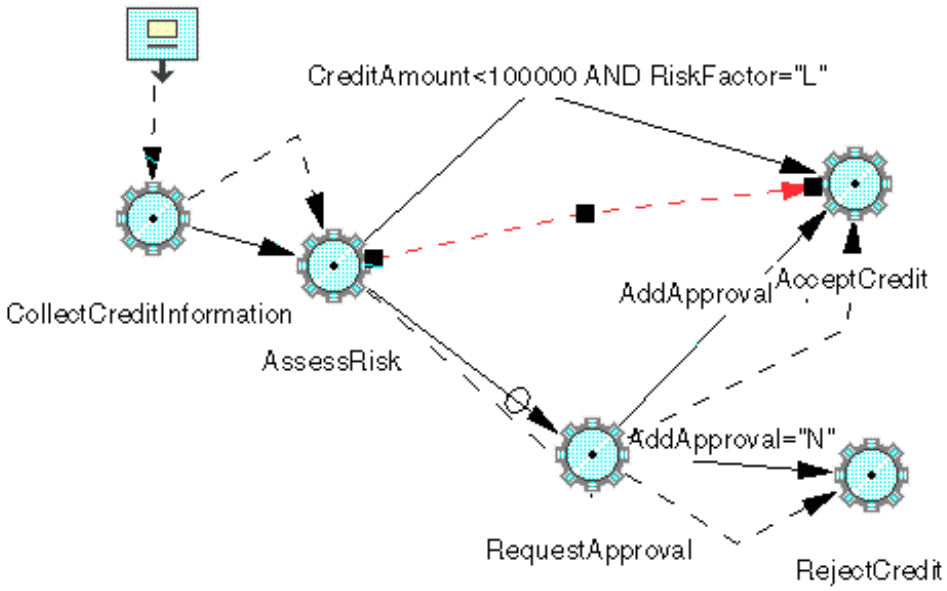

Figure 10.1: MQSeries Workflow: Sample Workflow Schema.

sition conditions of control flow connectors rather than by start conditions of workflow instances. Notice that the AssessRisk activity has CreditAmount and RiskFactor as output parameters. In an alternative form of data flow, a complex workflow can transfer data to its sub-workflows.

Dynamic adaptations of running workflow instances are not addressed in MQSeries Workflow. That system is based on the build-time versus run-time approach, discussed in Section 7.2. Hence, after the build time of a workflow, no changes or adaptations can be performed on workflow instances based on a given workflow schema. In MQSeries Workflow it is generally assumed that workflows do not change while they run, which is considered one property of so called production workflows [68].

The workflow language presented in this thesis is similar to the workflow language in MQSeries Workflow. Workflow schemas are represented by directed graphs, consisting of workflow schema nodes, which can be related to each other by control flow and data flow. Instead of transition conditions used in MQSeries Workflow, our approach uses start conditions. Start conditions are more adequate when it comes to workflow schema re-use, since different occurrences of workflow schemas can have different start conditions. Moreover, a given start condition can be used in different workflow schemas. Unlike our workflow language, MQSeries Workflow does not support recursion of workflow

schemas. The formalization presented in Section 3.2 considers both workflow schemas and workflow instances in a single formalism; since MQSeries Workflow relies on a build-time versus run-time approach, workflow schemas and workflow instances are treated separately, and they are not represented in a single formalism. As indicated above, the integrated formal specification of workflow schemas and workflow instances as proposed in Section 3.2 is well suited to design and implement controlled dynamic adaptations of running workflow instances based on formal correctness criteria, a functionality not aimed at by MQSeries Workflow.

Workflow Programming Languages

Workflow programming languages are used as internal representations of workflow specifications with the aim of exporting or importing them. As shown above, the MQSeries Workflow system presents a graphical interface to the user. In addition, there is an internal workflow language, called FDL (Flow Definition Language) which is used to import or export process models and to support the exchange of workflow models between different branches of an organization. Data types are defined, similar to record types in general purpose programming languages. There are syntactic constructs for program activities, implementing atomic workflows, and for control flow and data flow constraints. Fragments of the FDL specification of the sample workflow are shown in Figure 10.2.

In the Mobile system [12, 56], workflow schemas are specified using the Mobile script language. That project aims at supporting different workflow aspects in a modular way. This goal is reflected in the Mobile workflow language by supporting constructs for the definition of different workflow aspects. Besides the focus on workflow aspects, the Mobile workflow language provides extensibility. In particular, based on a set of predefined control flow operators, the user can define new control flow constructs to support the specific requirements of particular workflow applications. For instance, constructs to execute a set of activities in any sequential order can be specified. From an architectural point of view, each workflow aspect is covered by a server, dedicated to that particular aspect. The aim of this conceptual design and system architecture is to provide the system administrator with facilities and tools to use the aspects which are important for the particular workflow applications and to be able to extend the system with additional aspects, as they are required.

An incomplete specification of the sample workflow using the Mobile workflow language is shown in Figure 10.3. Each workflow aspect is represented by a language keyword and accompanying language constructs. Workflow schemas, control flow, and data flow are specified in sections, delimited by the respective keywords. The set of constructs

```
STRUCTURE 'CreditInfo'
    'CreditRequestor':  'PersonInfo';
    'Address':          STRING;
    'RiskFactor':       STRING;
    'AddApproval':      STRING;
    'CreditAmount':     LONG;
END 'CreditInfo'

PROCESS 'CreditRequest' ('PersonInfo')
    PROGRAM_ACTIVITY 'AcceptCredit' ('CreditInfo')
        PROGRAM 'NAcceptCredit'
        DONE_BY STARTER_OF_ACTIVITY 'CollectCreditInformation'
    END 'AcceptCredit'

    PROGRAM_ACTIVITY 'AssessRisk' ('CreditInfo', 'CreditInfo')
        PROGRAM 'NAssessCreditRisk'
        DONE_BY STARTER_OF_ACTIVITY 'CollectCreditInformation'
    END 'AssessRisk'

    PROGRAM_ACTIVITY 'CollectCreditInformation'
        ('PersonInfo', 'CreditInfo')
    PROGRAM_ACTIVITY 'RejectCredit' ('CreditInfo')
    PROGRAM_ACTIVITY 'RequestApproval'
        ('CreditInfo', 'CreditInfo')

    CONTROL FROM 'CollectCreditInformation' TO 'AssessRisk'
    CONTROL FROM 'AssessRisk' TO 'AcceptCredit'
        WHEN 'CreditAmount<100000'
    CONTROL FROM 'RequestApproval' TO 'RejectCredit'
        WHEN 'AddApproval="N"'
    CONTROL FROM 'RequestApproval' TO 'AcceptCredit'
        WHEN 'AddApproval="Y"'
    CONTROL FROM 'AssessRisk' TO 'RequestApproval'
        OTHERWISE
END 'CreditRequest'
```

Figure 10.2: Process Model specified in Flow Definition Language.

in this section includes sequential execution and branching. A complete workflow specification of the sample workflow can be found in [57].

```
WORKFLOW_TYPE CreditRequest (IN PersonInfo: CreditRequestor)
  /* definition of sub-workflows */
  WORKFLOW_DATA CreditInfo: c
  END_WORKFLOW_DATA

  CONTROL_FLOW
    sequence (CollectCreditInfo,
      sequence(AssessRisk,
      ifthen(c.CreditAmount<100000 AND c.RiskFactor == "L",
        AcceptCredit,
        sequence(RequestApproval, ifthen(c.AddApproval == "Y",
          AcceptCredit, RejectCredit)))))
  END_CONTROL_FLOW

  DATA_FLOW
    CreditRequest.CreditRequestor -> ci.CreditRequestor;
    AssessRisk.out_ci -> AcceptCredit.in_ci;
  END_DATA_FLOW
  /* definition of organizational aspect */
END_WORKFLOW_TYPE
```

Figure 10.3: Workflow Program specified in Mobile Workflow Language.

Workflow programming languages are used in two forms: (i) as intermediate formats to import or export workflow specifications (MQSeries Workflow) and (ii) as programming interfaces to workflow management systems (Mobile). In both cases, workflow modeling tools as font-ends of workflow management systems are necessary for a convenient development of workflow applications. Since in the system reported on in this thesis, workflow schemas are created using a workflow client application and workflow schema objects are stored as persistent objects using a persistency service, no intermediate format is required for workflow schemas. However, when importing workflow schemas developed with a commercial workflow management system will be required then the respective functionality to generate workflow schema objects from workflow programming code will be made available.

ADEPT

The ADEPT project at the University of Ulm aims at providing flexible workflow support for advanced applications, mainly in the context of hospital environments [82, 84].

Workflows are specified in the ADEPT workflow language, and dynamic modifications of workflows are modeled using ADEPT$_{flex}$, a formalism which consists of a set of rules, criteria, and methods which support the dynamic modification of workflow instances. This section provides a brief discussion of the ADEPT language, followed by an analysis of flexibility aspects in ADEPT$_{flex}$.

In the ADEPT workflow language, workflow schemas are specified as symmetric graphs, based on consistency rules defined in ADEPT. Graphs consist of different types of nodes and of different types of edges. In general, nodes represent tasks as well as control flow relevant information. For instance, there are special nodes for the start of a workflow and for the end of a workflow, called STARTFLOW and ENDFLOW, respectively. Every workflow starts with a dedicated STARTFLOW node and ends with a dedicated ENDFLOW node. This symmetric structure is a general property of workflows in ADEPT. There are branching nodes in workflow specifications, and there are join nodes. To keep workflow specifications symmetric, for each branching node there is a join node. Consistency is governed by compliance of workflow specifications to a set of predefined rules, for instance symmetry. Loops are modeled by other types of control flow relevant node types, called STARTLOOP and ENDLOOP. Each loop starts with a STARTLOOP node and ends with an ENDLOOP node.

Besides typed nodes, edges are also typed in ADEPT. In particular, there are control flow edges, and there are dedicated loop edges. Loop edges connect an ENDLOOP node to a STARTLOOP node. The ENDLOOP node is attached a predicate which is used to decide whether the loop is re-iterated. There are two forms of synchronization edges between workflow task nodes: soft synchronization edges and strict synchronization edges. A soft synchronization edge between a task node i and a task node j specifies that if i and j are executed then j can only start after i has terminated. On the other hand, a strict synchronization edge specifies that j can only start after i has terminated successfully. Examples which make use of these constructs to define constraints between concurrent tasks of a workflow are discussed in [84]. The information aspect is covered in ADEPT by variables which are global to a given workflow. Data flow is modeled indirectly by specifying accesses of tasks to variables. To avoid synchronization issues between concurrent branches of a workflow, two workflow tasks can access a data variable only if their execution is ordered, i.e., if they are performed sequentially.

Figure 10.4 displays the sample credit request workflow using the ADEPT workflow language. Notice that data and the flow of data is not modeled in this sample workflow specification. Due to the inherently asymmetric structure of the workflow, e.g., as shown in the previous representations based on directed graphs, the ADEPT representation requires additional modeling effort in order to transform the workflow specification into a

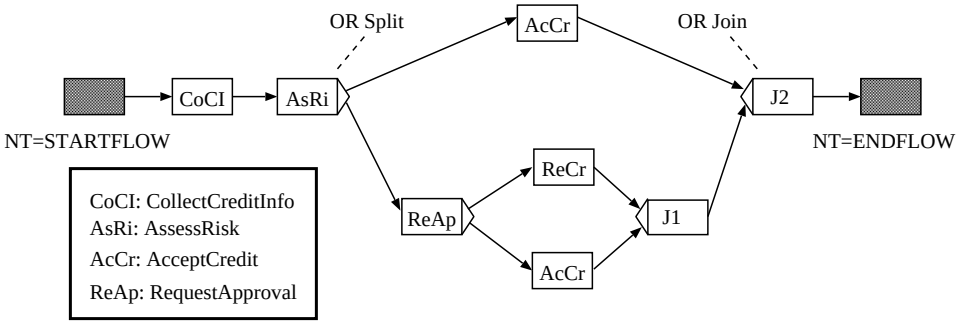

Figure 10.4: ADEPT: Sample Workflow Specification.

symmetric one. In Figure 10.4 two artificial join nodes are introduced ($J1$ and $J2$). In addition, the AcceptCredit workflow task appears twice in the workflow specification, in order to comply with the consistency criteria imposed by ADEPT. As can be seen in that rather simple example, the ADEPT workflow language imposes a fairly rigid structure on workflows, most prominently their symmetric structure. Hence, modeling a workflow with an inherent asymmetric structure requires additional nodes to be introduced. In complex asymmetric workflows, additional nodes in workflow specifications result in a rather complex and not easily maintainable nor understandable workflow specification.

Based on the ADEPT workflow language, adaptive workflows are addressed in [83]. An example from hospital environments motivates the need to dynamically modify workflow instances. Based on the ADEPT workflow language, a set of change operations are specified in the ADEPT$_{flex}$ framework [84]. In this framework, change operations to the structure of running workflows can be performed by users in a controlled manner. For instance, enhancing a workflow with a task involves the embedding of the added task into the workflow. The embedding of tasks in workflows is done by defining a set of tasks which have to be completed before the added one can start, and a set of tasks which can only be started after the added one has terminated.

After issuing operations on a workflow which change its structure, the consistency criteria may be violated. For instance, the workflow may loose its symmetric structure. Asymmetric workflows are considered incorrect in ADEPT. A set of transformation operations are proposed which transform the modified workflow in a symmetric one. The dynamic adaptation is correct if and only if this transformation is possible. Some of the limitations of ADEPT have already been discussed above; since ADEPT$_{flex}$ is based on ADEPT, these limitations also hold for the flexibility framework. In ADEPT$_{flex}$, ad-hoc changes of a single workflow instance as well as workflow type changes and the propagation of these changes to active workflow instance is addressed in [83].

Workflow Evolution

Flexibility issues in workflow management in general and workflow evolution in particular are investigated in [14] from a conceptual point of view, i.e., without a validation by an operational prototype. In that approach, workflow schemas consist of a set of tasks, which represent atomic workflow activities, i.e., workflow schemas are limited to one level of abstraction. There are control flow constraints between tasks, represented by directed edges between task nodes as well as specific fork and join nodes, representing branching and joining of execution control, respectively. Data flow is not modeled explicitly. In contrast, each workflow has a set of workflow variables, which are global to that workflow. Based on this workflow language, workflow schemas, workflow instances, and workflow enactment rules are defined. For each workflow schema at each point in time there may be multiple workflow instances.

Flexibility is provided by workflow evolution. The rational behind workflow evolution is similar to dynamic adaptations: Given a workflow instance with a workflow schema, adapt (or migrate) the workflow instance to a modified workflow schema. However, the approach is different to the one reported in this thesis. While the approach presented in [14] uses specific modification operations which are applied to the original workflow schema in order to modify it, the approach presented in this thesis is more general, since a workflow instance can be adapted to an arbitrary workflow schema, provided the consistency criteria are fulfilled. While we propose consistency criteria based on the workflow instance and the new workflow schema, the notion of compliance is introduced in [14], which defines which workflow instances can be migrated to the new workflow schema version, created by a set of modification operations, called workflow evolution primitives.

Workflow evolution primitives are partitioned in declaration primitives and flow primitives. While declaration primitives modify the declaration of workflow variables, flow primitives modify the control flow structure of workflow schemas. Typical declaration primitives are AddVar and RemoveVar (representing the adding and removing of a workflow variable, respectively), while AppendTask and RemoveTask are typical flow primitives. By applying these primitives to workflow schemas, the global variables, the task structure, and the control flow constraints of workflow schemas can be altered in an evolutionary way. Different policies to handle workflow evolution are discussed. Besides the most obvious ones (aborting all active workflow instances and completing all active ones with the original workflow schema), the progressive policies “migration to final workflow” and “migration to ad-hoc workflow” are proposed. In migration to final workflow, workflow instances which are compliant to the new workflow schema are migrated to that

schema, possibly after compensation activities, which are generally assumed to be available. In the migrate to ad-hoc workflow policy, the workflow administrator can choose to perform ad-hoc changes to the workflow instance, so that from an application-specific point of view, the workflow instance can now be migrated to the new workflow schema.

The work presented in [14] is based on a workflow language of rather limited expressiveness, since it does not support multi-level nesting of complex workflow schemas. Data flow between workflows is not modeled explicitly; the information aspect is covered by workflow variables, which are global within the context of a particular workflow. However, the interesting approach presents a formal model to specify workflow evolution in the presence of workflow schemas and multiple workflow instances, and different policies to handle workflow evolution from an organizational point of view are discussed.

Exceptions in workflow applications are addressed in [15]. In particular, the issue of providing support for designing exceptions which can be statically foreseen is tackled. The set of foreseen exceptions, specified as patterns, are maintained in a pattern catalog. Sample patterns are Remainder, Document Revision, and Termination. Patterns are specified using template structures as well as guidelines how to use the pattern. There is tool support for specifying exception patterns and for instantiating them in the context of particular workflow instances. Exception patterns can be regarded as application-oriented and often application-specific partial workflows, which are used to handle foreseen situations during workflow executions.

In the workflow management system introduced in this thesis, exception patterns can be defined as workflow schemas. By the capability to re-use workflow schemas in an arbitrary number of complex workflow schemas, exception patterns can be re-used. However, the specification of the patterns have to be defined by the workflow modeler. In addition to static workflow schema fragments, which are re-used, by dynamic adaptations, not only foreseen exceptions, but also unforeseen ones can be used in workflow instances. While dynamic adaptations currently are based on a new or modified workflow schema, and the modeling operations that have led to that workflow schema are not treated in detail, providing a library of partial workflow schemas describing exceptional situations will enhance the efficiency of exception handling. These issues will be investigated as future work.

Funsoft Nets

Funsoft nets are based on higher Petri nets and enhances them to incorporate different workflow aspects [38]; Funsoft nets are structured as follows: The node set is partitioned into places and transitions. Each place may include one or more typed data objects. Work-

flow activities are represented by transitions. Controlling workflow instances is done by passing documents and information between activities. The traditional Petri net formalism is enhanced with special constructs, e.g., there is a special form of transitions to represent alternative execution paths, represented by a "switch" transition.

The basic idea of this approach is the integration of different workflow aspects into a single formalism, namely Funsoft nets. This concept has implications on its usability. In particular, it allows to use a single formalism in different phases of the workflow application development process, i.e., Funsoft nets can be used during business process modeling, workflow modeling, and workflow execution. These nets provide a graphical notation to model data dependencies between tasks of application processes. In addition, organizational modeling is also supported by mapping role information to the net formalism. By providing appropriate means to specify external application programs to be used in the workflow executions, the operational aspect is also covered by this formalism. Besides modeling aspects, Funsoft nets can also be used as input for a workflow engine, i.e., they can also be used to control the execution of workflow instances. While the different workflow aspects are mapped into a single formalism, tools exist to provide views on certain aspects, for instance behavioral and operational. Nevertheless, the internal representation can become quite complex, which may lead to scalability problems in large workflow applications. However, the Funsoft net workflow language covers many interesting issues and is therefore discussed as an alternative approach to specifying workflows.

Figure 10.5 shows a simplified Funsoft net of the sample workflow. Places are represented by circles, and transitions are represented by rectangles. Each place may hold a set of typed data objects, for instance data object Credit Form. "Switch" transitions allow to model alternative execution paths. Assess Risk and Request Approval transitions have alternative outgoing edges to allow the explicit modeling of alternative branches, which are evaluated at runtime. The workflow starts with collecting the CollectCreditInfo activity, which takes a credit request as an input document and generates a credit form as an output document. The AssessRisk activity can either grant the credit request, in which case the granted request is transferred to the AcceptCredit activity, or it can postpone the decision, in which case a postponed credit request is transferred to the Request Approval activity. The workflow continues as is specified in Figure 10.5.

In [41] a flexibility mechanisms based on Funsoft nets is proposed. Simple dynamic modifications are supported, for instance to leave unspecified defined portions of the net to be filled when the workflow executes, denoted by late modeling. As indicated in [41], the functionality is implemented in a prototypical workflow management system.

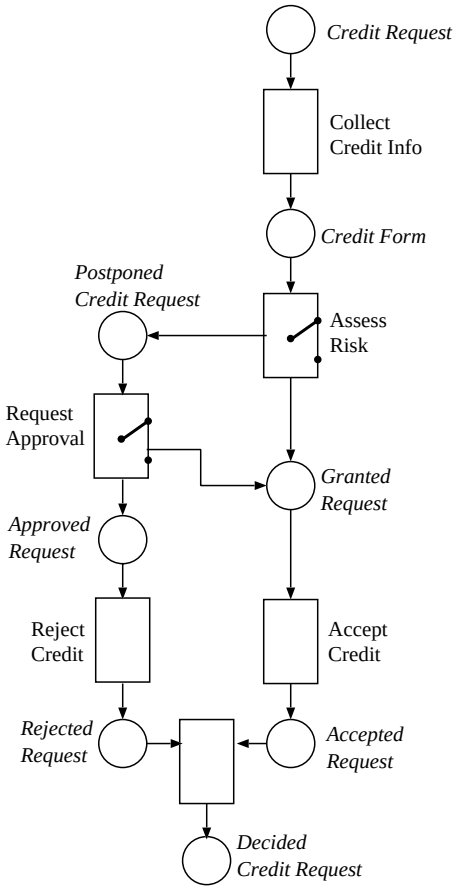

Figure 10.5: Funsoft Nets: Sample Workflow Model.

Other approaches

The issues of workflow flexibility and adaptability are also discussed in recent workshops [61, 101, 19]. The main contributions of these workshops are now briefly discussed. In [43], the constantly changing market environment of business processes is identified as a major motivation for adaptive workflow management. A taxonomy of adaptive workflow management is proposed, based on process level adaptations and resource level adaptations as requirements for a flexible workflow management system. Techniques for exception handling in workflow management systems are identified and classified in [69, 22].

An approach to enhance the flexibility of workflow management systems based on an integration of workflow and workspace management techniques is discussed in [58]. To classify flexibility requirements, *a priori* and *a posteriori* flexibility is characterized by properties of the application that are known before it starts and after it has started, respectively. In this approach, workflows are defined using tasks graphs. The execution semantics of task graphs are based on a state-chart variant and event condition action (ECA) rules. Tasks are considered as reactive components, whose behavior can be described by state-charts. By defining the semantics of state transitions using the ECA formalism, user defined control flow constructs can be introduced. This mechanism can be used to define less restrictive workflows. As an example of a less restrictive workflow, [59] provides mutual exclusion of different branches of a workflow. While this is an example of a user-defined control flow constraint, it is not less restrictive than the traditional approach but in fact more restrictive: Concurrent executions of the branches are not permitted. Instead the workflow tasks are forced to execute mutually exclusive. However, less restrictive workflow specifications can also be achieved with user-defined control flow constraints based on ECA rules. Dynamic changes of enacting workflow instances are also based on ECA rules. The execution of a change primitive generates an event, which is consumed by the respective workflow instances, which react to this event by a set of actions which implement that dynamic change. To control dynamic modifications, aborting, compensating and re-doing activities are taken into account. However, a formal foundation of correctness properties of dynamic changes is not provided.

In [44] and [113], approaches to adaptive workflow management based on Petri net formalisms are discussed. An open architecture for adaptive workflow management systems is introduced in [102]. Based on requirements for enterprise process management in the public administration, adaptive process support is characterized, and the SWATS architecture for adaptive workflow management is presented.

An interesting approach towards dynamic modifications of workflows is reported

in [74]. The basic idea of this approach is to provide a rule-based mechanism which allows for a given semantic exception to compute the set of workflow instances and the required modifications of these instances to cope with the exceptional situation. However, the exceptions and the respective rules to cope with the exceptions have to be known at workflow modeling time. Once a semantic exception occurs, a rule-based system decides whether certain workflow activities can be dropped or whether new workflow activities have to be started. In a second step, the set of workflow instances which are affected by that exception are determined. In a third step, for each affected workflow instance the respective region where changes have to be performed is determined. Finally, workflow modifications are processed to adjust the workflow instance to the exceptional situation, and the modified workflow instance is continued.

Besides workflow evolution and dynamic adaptation there are additional aspects of flexibility in workflow management systems, some of which are still under discussion. For example, flexible modeling of workflow schemas can also be regarded as a flexibility issue in workflow management. Flexibility in this context means to equip a workflow language with constructs which allow the flexible specification of complex properties of workflow schemas. An example of a complex control flow construct is the row construct, which is applied on a set of sub-workflows $1, 2, \ldots, n$. Assume there is a complex workflow k which is defined by $row(1, 2, \ldots, n)$. When a workflow instance based on that workflow schema is started, the sub-workflows $1, 2, \ldots, n$ can be executed in any order, however, sequentially. A workflow management system which supports the row construct guarantees that at each point in time at most one sub-workflow instance of the row construct is active. Finally, all sub-workflows of the complex workflow are completed, which terminates the complex workflow instance.

Another complex control flow construct is the “k out of n” construct. Consider a complex workflow schema k, consisting of n sub-workflows $1, 2, \ldots, n$. When a complex workflow instance based on this complex workflow schema is started, all n sub-workflow instances can be started in parallel. If and when k out of the n sub-workflow instances are completed then the other $(n - k)$ sub-workflows are no longer required an can therefore be disregarded. There are numerous other language constructs feasible, depending on the specific requirements of workflow applications. This situation has led to the definition of extendible workflow languages. However, one has to bear in mind that on the level of the workflow monitoring tool and on the workflow management system level, a great degree of complexity is introduced with these new language constructs. However, workflow applications with complex control structures can be specified in a more compact way using complex control flow constructs [56].

Along the lines of extending workflow meta schemas with additional properties, an

approach based on temporal aspects is introduced in [25]. Each workflow activity is assigned the expected duration as well as start and end times, and scheduling of workflow activities is based on the execution order of the workflow activities and on temporal aspects. This way, deadlines of workflow activities can be specified from an application-oriented point of view, and the workflow management system can make sure workflow activities are executed before their deadline is expired. Temporal aspects are not covered in the current version of the novel workflow management introduced. However, by adding the respective classes to the workflow meta schema and by providing the functionality in workflow objects, temporal aspects can be integrated in the conceptual design.

10.2 Object-Orientation and Distribution Aspects

As far as the technological background is concerned, the work presented in this thesis is related to the effort of the Object Management Group (OMG) to specify a Workflow Management Facility [78]. Today, the OMG has about 900 members, each of which has subscribed to supporting the CORBA middleware standard. After the specification of CORBA Services [80] can be considered complete, the OMG aims at specifying high-level services; these services are known as CORBA Facilities. In 1997, the Object Management Group issued a request for proposals for a workflow management facility [78]. There were responses from basically all major vendors of workflow management systems. In an analysis of the submissions, a number of deficiencies were identified, as far as compliance with the object paradigm is concerned. In addition, the embedding of the proposals in the CORBA context, i.e., the relationship of the proposals towards CORBA Services, was another point of criticism [95]. In a consolidation effort, a final proposal (called jointFlow [18]) was developed and adopted by the members of OMG. We remark that major workflow vendors most of which are also organized in the Workflow Management Coalition [118] proposed the jointFlow specification.

The workflow facility aims at two aspects: To provide runtime interoperability of existing workflow management systems and the ability to use workflow monitoring and auditing tools of different vendors in a given workflow application [91]. Since the specification is open to a variety of workflow languages and systems, it does not introduce a workflow language, and workflow schemas (or, in OMG terms, process definitions [18, 91]) are not discussed. However, paying attention to workflow schemas and workflow instances is important to support the re-use of workflow schemas and dynamic adaptations of workflow instances, two aspects, which are considered as important requirements for workflow support [91]. One reason for these limitations is probably the fact that the workflow ven-

dors, who played a key role in specifying the workflow management facility aim at a standard which their products can comply to with little or no re-implementation effort.

Since most workflow management systems in the market today were not designed and developed with object-oriented methods and tools, an obvious approach to provide run time interoperability is to build object wrappers around existing commercial workflow management systems using CORBA IDL. Calls to CORBA objects are then translated by the wrappers to native calls to specific workflow management system components. This approach results in two major issues: First, considerable overhead is introduced by the wrappers, which may lead to poor performance in large-scale workflow applications. Second, the vendor's workflow management systems are typically not implemented using CORBA Services. As an example, the specification states that persistency of workflow executions is provided by the workflow management systems, rather than by the CORBA persistency service. This approach to some extent contradicts the CORBA architecture, which specifies CORBA Services in order to use them to implement higher-level CORBA Facilities and, eventually, CORBA Applications.

The work presented in this thesis aims at different topics: CORBA is not used to provide a framework to support interoperability of existing workflow management systems. In contrast, the aim is to develop a workflow management system based on CORBA objects, and making extensive use of CORBA Services to implement the system. As a university research project, our effort is not restricted by the technical properties of existing products which we wanted to adapt to an upcoming standard. Hence, we could develop a consistent object model from scratch, which finally led to a novel workflow meta schema, as well as different variants of state-transition diagrams to describe the behavior of workflow instances. In a second step, CORBA was chosen as the technical infrastructure the system is based upon. During the project it turned out that CORBA Services indeed serve well in providing the basic functionality of a workflow management system. In particular, the persistency service, the life-cycle service, the relationship service, the trading service, and the event service proved important for the development, as explained in Section 8.2. It is interesting to notice that the list of requirements for workflow support as mentioned in [91] is well covered by the system presented in this thesis: (i) Support for changes in the underlying process model, (ii) Enable composition of re-usable business components, (iii) Allow for monitoring of process execution, (iv) Enable distribution of a process across business domains and (v) support assignment of process steps to workflow participants.

There are additional approaches to object-oriented workflow management. Three major ones are the Meteor$_2$ project at the University of Georgia [99], the METUFlow [24] project at the Middle East Technical University and the WorCOS project [96] at the Tech-

nical University of Dresden. In Meteor$_2$, workflow specifications are translated into executable code, which is then executed in a distributed fashion. While the specification tool allows complex workflows, i.e., supports nesting of workflows, the translation into executable code corresponds to a flattening of the nested structure. Hence, the resulting execution structure is flat. In contrast to the Meteor$_2$ approach, our system provides complex workflow instances, and the nested structure is present when the workflow executes. This is important when it comes to dynamic changes of running workflow instances, a feature which the Meteor$_2$ system does not aim at supporting. Each workflow instance has a task manager appended to it, which is responsible for starting and terminating the respective workflow instances. Like in our system, workflow control is distributed in the Meteor$_2$ system. This is achieved by task managers, which communicate to guarantee that workflow executions are controlled according to the respective workflow specifications. CORBA is used in the Meteor$_2$ system as a communication infrastructure of distributed workflow executions. One important application domain of the Meteor project is the health care sector [100].

In the METUFlow system [24], workflows are specified in an abstract way by their execution properties; the ACTA formalism [17] is used for expressing these properties. Workflows are executed in a distributed fashion in this approach, which is done by a guard concept [24]; guards are used to describe the behavior of concurrent systems (CSP [49]). The METUFlow project aims at exploiting this concept to control distributed workflow executions. In the METUFlow system, CORBA is used as a communication infrastructure, and the core workflow functionality is not based on CORBA COS. The WorCOS project aims at developing a workflow management system based on a CORBA infrastructure. The main emphasis of this project is the re-usability of workflow schemata and the integration of business objects in workflow applications.

The Mentor project at the University of the Saarland aims at providing persistent and distributed workflow executions [116]. In Mentor, workflow are specified using state-charts and activity charts, introduced in [45]. Control flow is specified by state-charts, which are composed of states and state transitions marked with ECA rules. A state transition can be performed if and when an event occurs and if then the condition evaluates to true. In this case, an action is performed. The functional decomposition of a workflow and data flow between functional entities is specified in activity charts. Distributed workflow executions are based on a partitioning of state-charts and activity charts. Based on a formal model, a complex state and its accompanying activity chart can be partitioned into a set of state-charts and activity charts [117]. It can be shown that the original representation and the partitioning are equivalent. On a technical level, the resulting workflow partitions are then executed in different sites of an organization. On the technical level, a

state-chart interpreter is used as a workflow engine [46]. Persistent message queues are used to guarantee that messages sent between sites to coordinate the execution of workflow partitions are never lost, and each message sent arrives at its destination site after finite time.

While the ADEPT project set out in the area of adaptive workflow management, recently additional topics are addressed, partitioning of workflow engines being one of them [4]. Workflows are specified in the ADEPT workflow specification language, as discussed above. Workflow specifications are then partitioned in workflow fragments, and each fragment is executed on a dedicated workflow server. Technically, migration of workflow instances between workflow servers is required to perform this functionality. This approach aims at reducing communication overhead between the workflow server and the workflow clients. This is important when communication links between workflow clients and the workflow server are comparatively slow, and when workflows are geographically distributed, a property which is likely to occur in enterprise-wide workflow applications. One issue in this context is dynamic role resolution. In dynamic role resolution — or dependent actor assignment [4] — agent information is computed only at the run time of the workflow, using parameter values which are only available when the workflow executes. As a result, the persons and even the departments involved in a particular workflow instance may not be known at modeling time. In terms of the workflow partitioning issue, this property means that the assignment of workflow fragments to workflow severs can only be done while the workflow executes. In general there is a trade off between migrating workflow instances between workflow servers (which of course means additional overhead in workflow execution control) and the reduction of communication overhead between workflow servers and workflow clients. The issues related to partitioning of workflow servers and dynamically adapting workflow instances are addressed in [85].

10.3 Workflow Management in Science and Engineering

While the main target domain of workflow management are business applications in commerce and public administration, workflow management concepts and systems are also used in other domains. In this section, workflow management approaches in science and in engineering are discussed. The use of computer science techniques in general and of process technologies in particular in the human genome project is discussed in [30]. Different scientific applications are overviewed in [54]. In scientific applications, workflow technology is typically embedded in working environments, which integrate a set of

domain-specific tools, much like the early idea of the generic WASA architecture. These environments include OASIS from UCLA [72] and the ZOO Experiment Management Environment from the University of Wisconsin [55, 3, 114, 39]. Application specific environments built on top of database systems are the Moby Dick system from Zurich University [88] or XBio [60]. Workflow approaches in scientific applications can also be identified in laboratory management systems, for instance the sister projects MapBase [36] and LabBase [103] developed at the Whitehead Institute and the MIT Center for Genome Research. Workflow management approaches in engineering include the CRISTAL project at CERN [70].

OASIS is an environment for data analysis, knowledge discovery, visualization, and collaboration, and is directed towards geo-applications. Its implementation is based on a CORBA-compliant distributed object management system. The OASIS project lays an emphasis on a loose cooperation of agents to achieve a common goal. ZOO is essentially a software package that allows scientists to manage experiments and data related to an experiment from a desktop machine through a uniform interface. However, ZOO emphasizes data modeling and experiment modeling at a high language level and at an individual basis, and workflow aspects are not in the center of attention.

LabBase also tries to bring workflow concepts into scientific applications. The related MapBase essentially is an object-oriented database based on the ObjectStore system for supporting human genome research; as part of the database schema, a so-called workflow module models experimental steps occurring in mapping procedures. Thus the system knows classes like Process and ProcessStep. LabBase is a system for managing workflows in large semi-automated laboratory projects which sits on top of the MapBase database system. Its workflow manager essentially controls the execution of laboratory protocols in which experiments to be conducted are described. Protocols may change frequently, since the sequencing or composition of their experiments are altered. The workflow manager is able to take care of this by taking appropriate input from the respective application at its interface; basically, it is programmed from outside for each experiment that is executed.

Finally, the CRISTAL system (Concurrent Repository and Information System for the Tracking of Assembly Lifecycles) emerged from the Compact Muon Solenoid experiment in particle physics [70]. The construction of a large-scale scientific apparatus used in particle physics at the European Laboratory for Particle Physics is performed in a complex process, which is distributed globally. In particular, the components are developed in a variety of laboratories in different continents. This process has to be monitored using workflow technology. An important aspect of this project is the integration of product data management tools, which are able to organize and control product data as well as product life-cycles.

10.4 Workflow Application Development Processes

While considerable work has been done in the area of software development processes where, for instance, the waterfall model [8], the fountain model [47], and the prototyping approach [11, 75] have been proposed, surprisingly little work has been published on workflow application development processes so far. However as was indicated in Chapter 4, workflow application development processes exhibit specific properties not found in traditional software development processes.

Kwan and Balasubramanian present a "Workflow-aware Information System Development Methodology" [64], which presents a rather high-level description of workflow application development processes. While that paper presents an interesting approach to the subject, there are some deficiencies. First, the paper neglects some important issues, e.g., it regards a workflow management system as given, and a system selection depending on survey data is not part of the development process. As motivated in Chapter 4, a system selection phase is considered an important requirement of a workflow methodology. A general approach for the development of workflow applications is discussed in [62]; that contribution concentrates on the phases of project planning, review and analysis, market survey and product selection, and user training and support. However, it does not present the internal structure and relationships between these phases in detail. The Workflow Management Coalition [65] defines required components for the development and usage of workflow applications. However, it does not define processes for the development of workflow applications.

Chapter 11

Conclusions and Future Work

This thesis presents our work on various research issues in advanced workflow management. Starting from workflow basics, in which workflow terminology was introduced, a workflow language based on directed graphs is specified using a mathematical formalization. That formalization of workflow schemas, workflow instances, and execution semantics of workflow instances are the foundation of the conceptual design of workflow management systems. From an application point of view, development methodologies for workflow applications are important to improve the planning and conduction of workflow projects, which may eventually lead to more efficient and more usable workflow applications.

Part II of this thesis presents realization concepts for workflow management systems. In particular, an object-oriented design is presented, which represents a workflow meta schema from a logical perspective. In the class diagram, workflow schemas and workflow instances are modeled as objects. This design decision has implications to dynamic adaptability and distributed workflow execution control. Dynamic adaptability of running workflow instances to new workflow schemas is achieved by changing the association of a workflow instance object with a workflow schema object. In addition, consistency criteria based on the notion of a valid mapping are used to check whether a given workflow instance can be adapted to a new workflow schema. The correctness criteria of dynamic adaptations utilizes the mathematical formalism of workflow schemas and workflow instances, proposed in Part I. To foster distributed workflow execution control, workflow objects are represented by information system objects. Workflow instance objects may reside in different sites of a distributed computing system. As far as the workflow execution control is concerned, workflow instance objects control their execution by sending and receiving messages. Instead of a centralized workflow engine which controls the execution of all workflow instances of an organization, workflow control is performed in a

fully distributed manner. By distributed workflow control, the risk of the central site to become a performance bottleneck is reduced, and fault tolerance of workflow instances is increased. However, efficient distributed middleware is an important prerequisite to gain performance in this context.

In the implementation level, the distributed, object-oriented middleware CORBA is used. In particular, workflow objects including workflow schema and workflow instance objects as well as accompanying objects are represented by CORBA objects. As a foundation, a set of CORBA Services are designed, implemented, and used. In particular, a persistency service makes sure workflow objects are generally persistent, a life-cycle service is responsible for creating, moving, and deleting workflow objects, a trading service allows to locate objects in distributed environments according to specific properties of objects. An event service manages events that occur during workflow executions by the use of event channels. Complex relationships between objects is maintained by a relationship service. The encapsulation of the specification of the services and their implementation simplifies the maintenance of the system. For example, in the course of the project the implementation of the persistency service was changed from a relational database system to a POSIX file system without changing the higher-level objects such as workflow objects or the workflow client application.

Research results tend to open the door for new research issues. To this end, the use of compensation techniques is an interesting area for future work. Compensation has been an important concept in advanced transaction models, for instance in Sagas [31]. A Saga consists of a set of sub-transactions, each of which has a compensating sub-transaction associated to it, which semantically undoes the sub-transaction it is assigned to. For instance, the booking of a flight can be semantically undone by a reversal. Each sub-transaction of a Saga is executed as an atomic transaction. However, the sub-transactions of a given Saga are either executed successfully, or, if one sub-transactions fails, all sub-transactions which have already been executed successfully are undone, using compensating sub-transactions.

Using the compensation concept, the applicability of dynamic adaptations in workflow management systems can be broadened. As explained above, a workflow instance can be dynamically adapted to a new workflow schema if there is a valid mapping from the workflow instance to the workflow schema. This mapping is based on the workflow instance nodes which already have been executed. Using compensation of workflow instance nodes would allow to roll back the workflow instance under consideration. Rolling back workflow instance nodes which cannot be mapped to the new workflow schema node allows to perform the dynamic adaptation in cases for which the dynamic adapta-

tion was not permitted without compensation. While the workflow management system presented in this thesis provides the basic properties to perform this new functionality, the formal characterization of dynamic adaptability and, of course, the implementation and the graphical user interface will require additional work.

Another area of future work is business objects and component-based software development and the relationship of these aspects with workflow management. As indicated earlier, business objects can be used by workflow applications in two ways: First, data flow can be in fact business object flow, in the sense that business objects are transferred between workflow instances. Second, the functionality provided by business objects can be used to perform atomic workflows. This approach provides a new way of polymorphism in workflow management, since the operations that are implemented by an atomic workflow instance depend on the particular business object received by data flow. In a simple example, to display the values of a business object in an atomic workflow, the display() method of that business object can be invoked. Hence, the workflow can display arbitrarily structured business objects, since the implementation of the workflow, i.e., its operational aspect, is attached to the business objects. While the functionality is already designed and implemented, with the advent of commercial business objects the stability and performance of the integration will be a future topic.

As far as the re-use is concerned, the object level seems to have too fine granularity for re-use in complex business applications, since in order to re-use required functionality, numerous objects have to be re-used, while new objects need to be created in order to provide the needs required by the application. In this context, frameworks and business components are in the center of attention. Frameworks define the structure and the behavior of objects and of collections of objects, which semantically belong together. By providing specific extension points, the positions in which additional code is added to complement the functionality is specified, leading to easier and more convenient software development. In this context, the use of frameworks in workflow applications is an interesting area of future work. The approach taken by the workflow management facility as specified by the Object Management Group regards a workflow management system as a component. By defining the interface of workflow management systems in CORBA interface definition language, interoperability of workflow components, i.e., interoperability of workflow management systems of different vendors, can be achieved. However, we believe that workflow components can also be defined on a finer granularity. For instance, complex workflow schemas and complex workflow instances can also be regarded as components. In the workflow management system presented in this thesis, this approach makes sense, since workflow execution control is performed by workflow instances. However, future work is required in this context to properly specify the struc-

ture and behavior of workflow components, as well as operations to combine workflow components to new workflow components.

In the middleware layer, CORBA is a standard as far as specification is concerned. However, a strategic alliance of influential software vendors are aiming at specifying another distributed object middleware standard based on the Java programming language, called Enterprise Java Beans. As of today, there is no complete adopted standard for Enterprise Java Beans. However, there are some implementations already, called application servers. In a recent development, Object Management Group is trying to combine the CORBA architecture with Enterprise Java Beans. Time will tell which of the standards in distributed object management will become the de-facto standard. A nice property of the layered architecture of the workflow management system reported on in this thesis is the separation of levels. This means that the lower level of the implementation (CORBA Services, in the recent prototypical implementation) can be substituted by another implementation, for instance Enterprise Java Beans. Since the workflow functionality is completely implemented in the middle layer, the facilities level, the changes in the lower level are transparent for workflow objects and, of course, the workflow client application.

Bibliography

[1] Aho, A.V., Ullman, J.D.: *Foundations of Computer Science.* New York: Computer Science Press 1992

[2] Alonso, G., Fiedler, U., Hagen, C., Lazcano, A., Schuldt, H., Weiler, N.: *WISE: Business-to-Business E-Commerce.* In: Proc. 9^{th} International Workshop on Research Issues in Data Engineering. Information Technology for Virtual Enterprises (RIDE-VE'99), pp 132–139. March 1999

[3] Anjur, V., Ioannidis, Y., Livny, M.: *FROG and TURTLE: Visual Bridges Between Files and Object-Oriented Data*. Proc. 8th International Conference on Scientific and Statistical Database Management 1996, pp 76–85. Los Alamitos: IEEE Computer Society Press 1996

[4] Bauer, T., Dadam, P.: *Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows.* In Dadam, Reichert (Editors): Proc. Informatik'99 Workshop on Enterprise-Wide and Cross-Enterprise Workflow Management: Concepts, Systems, Applications. Ulmer Informatik-Berichte Nr. 99-07, pp 25–32. Universität Ulm 1999

[5] Bauzer Medeiros, C., Vossen, G., Weske, M.: *WASA: A Workflow-Based Architecture to Support Scientific Database Applications*. Revell, Tjoa (Editors): Proc. 6th DEXA Conference, Springer LNCS 978, pp 574–583. Berlin: Springer 1995

[6] Bauzer Medeiros, C., Vossen, G., Weske, M.: *GEO-WASA: Supporting Geoprocessing Applications using Workflow Management*. Proc. 7th Israeli Conference on Computer Systems and Software Engineering, pp 129–139. Los Alamitos: IEEE Computer Society Press 1996

[7] Bernstein, P.A., Hadzilacos, V., and Goodman, N.: *Concurrency Control and Recovery in Database Systems*. Reading: Addison Wesley 1987

[8] Boehm, B.: *Software-Engineering Economics*. Englewood Cliffs: Prentice Hall 1981

[9] Bohrer, K., Johnson, V., Nilsson, A., Rubin, B.: *Business Process Components for Distributed Object Applications*. Communications of the ACM, pp 43–48, Vol 41 No 6. New York: ACM 1998

[10] Bohrer, K.: *Architecture of the San Francisco Framework.* IBM Systems Journal Vol 37, No 2, 1998

[11] Budde, R.: *Prototyping. An Approach to Evolutionary Systems Development*. Berlin: Springer 1992

[12] Bußler, C., Jablonski, S., Schuster, H.: *A New Generation of Workflow Management Systems: Beyond Taylorism with MOBILE*. pp 17–20. ACM SIGOIS Bulletin 17(1) 1996

[13] Casanave, C.: *Business-Object Architectures and Standards*. Proc. OOPSLA '95, Workshop on Business Objects Design and Implementation, pp 7–28. London: Springer 1995

[14] Casati, F., Ceri, S., Pernici, B., Pozzi, G.: *Workflow Evolution*. Data and Knowledge Engineering Vol 24 No 3 1998. pp 211–238. Elsevier Science 1998

[15] Casati, F., Fugini, M.G., Mirbel, I.: *An Environment for Designing Exceptions in Workflows.* Proc. CAiSE'98. pp 139–157. Springer Lecture Notes in Computer Science 1413. Berlin: Springer 1998

[16] Chen, P.P.S.: *The Entity-Relationship Model – Toward a Unified View of Data.* ACM Transactions on Database Systems Volume 1, pp 9–36. ACM 1976

[17] Chrysanthis, P.K., Ramamritham, K.: *ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behaviour*. Proc. 1990 ACM SIGMOD Conference on Management of Data, pp 194–203. New York: ACM Press 1990

[18] CoCreate Software, Concentus, CSE Systems, Data Access Technologies, Digital Equipment Corp., DSTC, EDS, FileNet Corp., Fujitsu Ltd., Hitachi Ltd., Genesis Development Corp., IBM Corp., ICL Enterprises, NIIIP Consortium, Oracle Corp., Plexus - Division of BankTec, Siemens Nixdorf Informationssysteme, SSA, Xerox: *BODTF-RFP 2 Submission Workflow Management Facility (jointFlow)*. OMG Document bom/98-06-07 1998

[19] Dadam, P., Reichert, M. (Editors): *Proceedings Informatik'99 Workshop on Enterprise-Wide and Cross-Enterprise Workflow Management: Concepts, Systems, Applications.* Ulmer Informatik-Berichte Nr. 99-07. Universität Ulm 1999

[20] Damschik, I., Häntschel, I.: *Evaluation of Workflow Systems.* (in German) Wirtschaftsinformatik, Vol 37 No 1, pp 18–23. Wiesbaden: Vieweg 1995

[21] Data Access Technologies, Electronic Data Systems, NIIIP, Sematech, Genesis Development Corp., Prism Technologies, Iona Technologies, Inc.: *OMG Business Object Domain Task Force BODTF-RFP 1 Submission: Combined Business Object Facility – Business Object Architecture* OMG Document bom/97-11-09 1997

[22] Deiters, W., Goesmann, T., Just-Hahn. K., Löffeler, T. Rolles, R.: *Support for Exception Handling Through Workflow Management Systems.* Proc. CSCW-98 Workshop: Towards Adaptive Workflow Systems. (download from http://ccs.mit.edu/klein/cscw98 on 09-24-1998)

[23] Derungs, M., Vogler, P., Österle, H.: *Criteria Catalog Workflow Systems.* (in German) Technical Report IM HSG/CC PSI/1, Hochschule St. Gallen fr Wirtschafts-, Rechts- und Sozialwissenschaften: Institut für Wirtschaftsinformatik 1995

[24] Dogac, A., Gokkoca, E., Arpinar, S., Koksal, P., Cingil, I., Arpinar, B., Tatbul, N., Karagoz, P. Halici, U., Altinel, M.: *Design and Implementation of a Distributed Workflow Management System: METUFlow*. Workflow Management Systems and Interoperability. Nato ASI Series, Series F: Computer and Systems Sciences, Vol. 164, 61–91. Berlin: Springer 1998

[25] Eder, J., Panagos, E., Rabinovic, M.: *Time Constraints in Workflow Systems.* Jarke, Oberweis (Editors): Proc. 11th International Conference on Advanced Information Systems Engineering (CAiSE'99), pp 286–300. Springer Lecture Notes in Computer Science 1626, Berlin: Springer 1999

[26] Electronic Data Systems, Data Access Technologies, Genesis Development Corp., NIIIP, System Software Associates, Inc.: *OMG Business Object Domain Task Force BODTF-RFP 1 Submission: Combined Business Object Facility.* OMG Document bom/98-05-03 1998

[27] Ellis, C., Keddara, K., Rozenberg, G.: *Dynamic Change Within Workflow Systems.* Proc. Conference on Organizational Computing Systems, pp 10–22. Milpitas 1995

[28] Focke, B.: *Design and Implementation of a Graphical Specification Tool for Workflow Management Systems.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1997

[29] Fowler, M.: *Analysis Patterns: Reusable Object Models.* Reading: Addison-Wesley 1997

[30] K.A. Frenkel. *The Human Genome Project and Informatics*. Communications of the ACM, pp 41–51, Vol 34 No 11. New York: ACM 1991

[31] Garcia-Molina, H., Salem, K.: *Sagas.* In Proc. ACM SIGMOD Conference on Management of Data. pp 249–259. New York: ACM Press 1987

[32] Georgakopoulos, D., Hornick, M., Sheth, A.: *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. Distributed and Parallel Databases Vol 3, pp 119–153. Boston: Kluwer 1995

[33] Geppert, A., Tombros, D., Dittrich, K.R.: *Defining the Semantics of Reactive Components in Event-Driven Workflow Execution with Event Histories.* Information Systems Vol 23 No 3, pp 235–252. Elsevier Science 1998

[34] Gillmann, M., Weissenfels, J., Weikum, G., Kraiss, A.: *Performance Assessment and Configuration of Enterprise-Wide Workflow Management Systems.* Dadam, Reichert (Editors): Proc. Informatik'99 Workshop on Enterprise-Wide and Cross-Enterprise Workflow Management: Concepts, Systems, Applications. Ulmer Informatik-Berichte Nr. 99-07, pp 18–24. Universität Ulm 1999

[35] Goesmann, T., Striemer, R.: *Development of Workflow Applications in Commercial Settings: Experiences and Consequences.* (in German) Technical Report ISST-Bericht 44/98, Fraunhofer ISST 1998

[36] Goodman, N.: *An Object-Oriented DBMS War Story: Developing a Genome Mapping Database in C++*. W. Kim (ed.), *Modern Database Systems — The Object Model, Interoperability, and Beyond*, pp 216–237. Reading: Addison-Wesley 1995

[37] Gray, J., Reuter, A.: *Transaction Processing: Concepts and Techniques.* San Mateo: Morgan Kaufmann 1993

[38] Gruhn, V. *Validation and Verification of Software Process Models.* Ph.D. Thesis. Universität Dortmund 1991

[39] Haber, E., Ioannidis, Y.E., Livny, M.: *Opossum: Desktop Schema Management through Customizable Visualization*. Proc. 21st International Conference on Very Large Data Bases, pp 527–538. Zurich 1995

[40] Hacker, M.: *Process Controlling and Auditing in CORBA-Based Workflow Management Systems.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1998

[41] Hagemeyer, J., Herrmann, T., Just-Hahn, K., Striemer, R.: *Flexibility in Workflow Management Systems.* (in German) Software-Ergonomie '97, pp 179–190, Dresden, March 1997

[42] Hammer, M., Champy, J.: *Re-Engineering the Corporation. A Manifesto for Business Revolution.* Addison-Wesley 1994

[43] Han, Y., Sheth, A., Bußler, C.: *A Taxonomy of Adaptive Workflow Management.* Proc. CSCW'98 Workshop: Towards Adaptive Workflow Systems. (download from http://ccs.mit.edu/klein/cscw98/ on 09-24-1998)

[44] Han, Y., Weber, H.: *Adaptive Workflow and Software Architecture Thereof*. Journal of Integrated Design and Process Science 2(2) pp 1–21. Austin: Society for Design and Process Science 1998

[45] Harel, D. *On Visual Formalisms*. Communications of the ACM (31) 1988, pp 514–530. New York: ACM 1988

[46] Harel, D., Politi, M.: *Modeling Reactive Systems with Statecharts: The Statemate Approach.* Part No. D-1100-43, i-Logix Inc., Andover, MA 01810, 1996

[47] Henderson-Sellers, B., Edwards, J.M.: *The Object-Oriented Systems Life Cycle*. Communications of the ACM 33(9) 1990, pp 142–159. New York: ACM 1990

[48] Herrmann, T., Scheer, A.-W., Weber, H. (Editors): *Improvement of Business Processes with Flexible Workflow Management Systems.* (in German) Volume 1, Berlin: Physica 1998

[49] Hoare, C.A.R.: *Communicating Sequential Processes.* Englewood Cliffs: Prentice Hall 1985

[50] Holten, R., Striemer, R., Weske, M.: *Comparing Approaches to Develop Workflow Applications (in German).* Oberweis, Sneed (Editors): Software Management 1997, 258–274

[51] Hündling, J.: *Development of a Graphical User Interface for a CORBA-Based, Flexible Workflow Management System.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1998

[52] IBM. *IBM FlowMark: Modeling Workflow, Version 2 Release 2*. Publ. No SH-19-8241-01, 1996

[53] IBM. *IBM MQSeries Workflow: Concepts and Architecture, Version 3.2*. Publ. No GH12-6285-01, 1999

[54] Ioannidis, Y. (Editor): *Special Issue on Scientific Databases*. Data Engineering Bulletin 16 (1) 1993

[55] Ioannidis, Y., Livny, M., Gupta, S., Ponnekanti, N.: *ZOO: A Desktop Experiment Management Environment*. Proc. 22nd International Conference on Very Large Data Bases 1996, pp 274–285

[56] Jablonski, S., Bußler, C.: *Workflow-Management: Modeling Concepts, Architecture and Implementation* International Thomson Computer Press 1996

[57] Jablonski, S., M. Böhm, W. Schulze. (Editors) *Workflow Management: Development of Applications and Systems.* (in German) Heidelberg: dpunkt 1997

[58] Joeris, G.: *Aspects and Concepts of Flexibility in Workflow Management Systems.* (in German) Proc. D-CSCW98 Workshop on Flexibility and Cooperation in Workflow Management Systems, pp 3–12. Technical Report Angewandte Mathematik und Informatik 18/98-I, Universität Münster 1998

[59] Joeris, G.: *Defining Flexible Workflow Execution Behaviors.* Dadam, Reichert (Editors): Proc. Informatik'99 Workshop on Enterprise-Wide and Cross-Enterprise Workflow Management: Concepts, Systems, Applications. Ulmer Informatik-Berichte Nr. 99-07, pp 49–55. Universität Ulm 1999

[60] Kamel, N., Song, T., Kamel, M.: *An Approach for Building an Integrated Environment for Molecular Biology Databases.* Distributed and Parallel Databases (1) 1993, pp 303–327. Boston: Kluwer 1993

[61] Klein, M. (Editor) *Towards Adaptive Workflow Systems.* Workshop in The 1998 ACM Conference on Computer Supported Cooperative Work (Online Proceedings at http://ccs.mit.edu/klein/cscw98/ on Sept 09-24-98)

[62] Kobielus, J. G.: *Workflow Strategies.* Foster City: IDG 1997

[63] Kuropka, D.: *Specification and Implementation of a CORBA-Based Persistent Workflow Engine.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1998

[64] Kwan, M., Balasubramanian, P.R.: *Adding Workflow Analysis Techniques to the IS Development Toolkit.* Proc. 31st Hawaii International Conference on System Sciences, Track on Internet and the Digital Economy (Vol IV), 312–321. IEEE Computer Society Press 1998

[65] Lawrence, P. (Editor): *Workflow Handbook 1997.* Chichester: John Wiley 1997

[66] Lewandowski, S.M.: *Frameworks for Component-Based Client/Server Computing.* ACM Computing Surveys Vol 30 No 1 1998, pp 3–27. New York: ACM 1998

[67] Leymann, F., Altenhuber, W.: *Managing Business Processes as an Information Resource*. pp 326–347. IBM Systems Journal 33 1994

[68] Leymann, F., Roller, D.: *Production Workflow: Concepts and Techniques*. Upper Saddle River: Prentice Hall 2000

[69] Luo, Z., Sheth, A.: *Defeasible Workflow, its Computation and Exception Handling.* Proc CSCW'98 Workshop: Towards Adaptive Workflow Systems. (download from http://ccs.mit.edu/klein/cscw98/ on 09-24-1998)

[70] McClatchey, R., Baker, N., Harris, W., Le Goff, J.-M., Kovacs, Z., Estrella, F., Bazan, A., Le Flour, T.: *Version Management in a Distributed Workflow Application.* Proc. 8th International Workshop on Database and Expert Systems Applications 1997, pp 10–15. Los Alamitos: IEEE Computer Society Press 1997

[71] Meidanis, J., Vossen, G., Weske, M.: *Using Workflow Management in DNA Sequencing*. Proc. 1st IFCIS International Conference on Cooperative Information Systems (CoopIS) 1996, pp 114–123. Los Alamitos: IEEE Computer Society Press 1996

[72] Mesrobian, E., Muntz, R., Shek, E., Nittel, S., LaRouche, M., Kriguer, M.: *OASIS: An Open Architecture Scientific Information Systems*. Proc. 6th International Workshop on Research Issues in Data Engineering 1996, pp 107–116

[73] Milner, R. *A Calculus of Communicating Systems.* Springer LNCS 92. Berlin: Springer 1980

[74] Müller, R., Rahm. E.: *Rule-Based Dynamic Modification of Workflows in a Medical Domain.* Buchmann (Editor): Proc. of BTW'99. Informatik Aktuell, pp 429–448. Berlin: Springer 1999

[75] Mullin, M.: *Rapid Prototyping for Object-Oriented Systems*. Reading: Addison-Wesley 1990

[76] Object Management Group: *The Common Object Request Broker: Architecture and Specification.* OMG Document 91-12-1 1991

[77] Object Management Group: *Common Facilities RFP-4: Common Business Objects and Business Object Facility*. OMG Document 96-01-04 1996

[78] Object Management Group: *Workflow Management Facility: Request for Proposals.* OMG Document cf/97-05-06 1997

[79] Object Management Group: *Unified Modeling Language Specification. Version 1.3*, March 1999 (download from www.rational.com on 15-07-1999)

[80] Object Management Group: *CorbaServices: Common Object Services Specification*. OMG Document 98-12-09 1998

[81] O'Neil, P.: *Database: Principles, Programming, Performance.* San Francisco: Morgan Kaufmann 1994

[82] Reichert, M., Dadam, P.: *A Framework for Dynamic Changes in Workflow Management Systems.* Proc. 8th International Workshop on Database and Expert Systems Applications 1997, pp 42–48. Los Alamitos: IEEE Computer Society Press 1997

[83] Reichert, M., Hensinger, C., Dadam, P.: *Supporting Adaptive Workflows in Advanced Application Environments*. EDBT Workshop on Workflow Management Systems. Valencia, March 1998

[84] Reichert, M., Dadam, P.: *Supporting Dynamic Changes of Workflows Without Loosing Control.* Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Vol. 10, No. 2, 1998

[85] Reichert, M., Bauer, T., Dadam, P.: *Enterprise-Wide and Cross-Enterprise Workflow Management: Challenges and Research Issues for Adaptive Workflows.* Dadam, Reichert (Editors): Proc. Informatik'99 Workshop on Enterprise-Wide

and Cross-Enterprise Workflow Management: Concepts, Systems, Applications. Ulmer Informatik-Berichte Nr. 99-07, pp 56–64. Universität Ulm 1999

[86] Reuß, T.: *Process Modeling in Laboratory Environments.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1997

[87] Reuß, T., Vossen, G., Weske, M.: *Modeling Samples Processing in Laboratory Environments as Scientific Workflows.* Proc. 8th International Workshop on Database and Expert Systems Applications 1997, pp 49–55. Los Alamitos: IEEE Computer Society Press 1997

[88] Rieche, B., Dittrich, K.: *A Federated DBMS-Based Integrated Environment for Molecular Biology*. Proc. 7th International Working Conference on Scientific and Statistical Database Management 1994, pp 118–127

[89] Rusinkiewicz, M., Sheth, A.: *Specification and Execution of Transactional Workflows*. Won, K. (Editor): Modern Database Systems The Object Model, Interoperability, and Beyond. pp 592–620. ACM Press 1995

[90] Cichocki, A., Rusinkiewicz, M.: *Migrating Workflows.* Dogac, Kalinichenko, Özsu, Sheth (Editors): Workflow Management Systems and Interoperability. NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 164, pp 339–355. Berlin: Springer 1998

[91] Schmidt, M.-T.: *Building Workflow Business Objects.* OOPSLA'98 Workshop 8: Business Object Design and Implementation IV: From Business Objects to Complex Adaptive Systems (download from www.jeffsutherland.org/oopsla98/mts.html on 11-11-98)

[92] Schuschel, H.: *Flexibility Issues in a Distributed, Flexible Workflow Management System.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1998

[93] Schuldt, H., Alsonso, G., Schek, H.-J.: *Concurrence Control and Recovery in Transactional Process Management.* Proc. ACM Symposium on Principles of Database Systems (PODS'99), pp 316–326. New York: ACM Press 1999

[94] Schuldt, H., Popovici, A., Schek, H.-J.: *Give me all I pay for — Execution Guarantees in Electronic Commerce Payment Processes.* Dadam, Reichert (Editors):

Proc. Informatik'99 Workshop on Enterprise-Wide and Cross-Enterprise Workflow Management: Concepts, Systems, Applications. Ulmer Informatik-Berichte Nr. 99-07, pp 10–17. Universität Ulm 1999

[95] Schulze, W.: *Evaluation of the Submissions to the Workflow Management Facility RFP.* OMG Document bom/97-09-02 1997

[96] Schulze, W.: *Object-oriented Implementation Techniques for Workflow Management Systems in OMA-compliant Architectures.* (in German) Jablonski, S., Böhm, M., Schulze, W. (Editors): Workflow-Management: Entwicklung von Anwendungen und Systemen. Heidelberg: dpunkt 1997

[97] Serries, T.: *Distribution and Scalability in CORBA-Based Workflow Management Systems.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1998

[98] Sheth, A., Georgakopoulos, D., Joosten, S.M.M., Rusinkiewicz, M., Scacchi, W., Wileden, J., Wolf, A.: *Report from the NSF Workshop on Workflow and Process Automation in Information Systems*. Technical Report UGA-CS-TR-96-003. University of Georgia 1996

[99] Sheth, A., Kochut, K.: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems.* Dogac, Kalinichenko, Özsu, Sheth (Editors): Workflow Management Systems and Interoperability. NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 164, pp 35–60. Berlin: Springer 1998

[100] Sheth, A., Kochut, K., Miller, J., Worah, D., Das, S., Lin, C.: *Supporting Statewide Immunization Tracking using Multi-Paradigm Workflow Technology*. Technical Report UGA-CS-TR-96-001. University of Georgia 1996

[101] Siebert, R., Weske, M. (Editors): *Flexibility and Cooperation in Workflow Management Systems.* (in German) Workshop at D-CSCW98, German Conference on CSCW 1998. Technical Report Angewandte Mathematik und Informatik 18/98-I, Universität Münster 1998

[102] Siebert, R.: *An Open Architecture for Adaptive Workflow Management Systems*. Özsu, Dogac, Ulusoy (Editors): Proc. Conference on Integrated Design and Process Technology IDPT-Vol.2, pp 79–85. Austin: Society for Design and Process Science 1998

[103] Stein, L., Rozen, S., Goodman, N.: *Managing Laboratory Workflow with LabBase.* Proc. 1994 Conference on Computers in Medicine

[104] Sutherland, J.: *The Object Technology Architecture: Business Objects for Corporate Information Systems.* Proc. OOPSLA'95 Workshop: Business Object Design and Implementation. Berlin: Springer 1997

[105] Vossen, G., Weske, M.: *The WASA Approach to Workflow Management for Scientific Applications*. Dogac, Kalinichenko, Özsu, Sheth (Editors): Workflow Management Systems and Interoperability. NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 164, pp 145–164. Berlin: Springer 1998

[106] Vossen, G., Weske, M.: *The WASA2 Object-Oriented Workflow Management System*. Proc. 1999 ACM SIGMOD Conference on Management of Data, pp 587–589. New York: ACM 1999

[107] Vossen, G.: *Data Models, Database Languages, and Database Management Systems.* (in German) 3^{rd} Edition. München: Oldenbourg 1999

[108] Weske, M., Vossen, G.: *Workflow Languages*. Bernus, Mertins, Schmidt (Editors): Handbook on Architectures of Information Systems. (International Handbooks on Information Systems), pp 359–379. Berlin: Springer 1998

[109] Weske, M.: *Flexible Modeling and Execution of Workflow Activities.* Proc. of 31st Hawai'i International Conference on System Sciences, Software Technology Track (Vol VII), 713–722. IEEE Computer Society Press 1998

[110] Weske, M., Goesmann T., Holten, R., Striemer, R.: *A Reference Model for Workflow Application Development Processes.* Georgakopoulos, Prinz, Wolf (Editors) Proc. International Joint Conference on Work Activities Coordination and Collaboration, pp 1–10. ACM 1999

[111] Weske, M., Vossen, G., Bauzer Medeiros, C., Pires, F.: *Workflow Management in Geoprocessing Applications*. Proc. 6th ACM International Symposium on Geographic Information Systems (ACM-GIS'98), pp 88–93. New York: ACM 1998

[112] Weske, M.: *Business Object: Concepts, Architectures, Standards* (in German) Wirtschaftsinformatik Vol 41 No 1 1999, pp 4–11. Wiesbaden: Vieweg 1999

[113] White, G.M.: *Towards the Analysis and Synthesis of Adaptive Workflow Systems.* Proc. CSCW-98 Workshop: Towards Adaptive Workflow Systems. (download from http://ccs.mit.edu/klein/cscw98 on 09-24-1998)

[114] Wiener, J.L., Ioannidis, Y.E.: *A Moose and a Fox Can Aid Scientists with Data Management Problems*. Proc. 4th International Workshop on Database Programming Languages, pp 376–398 1993

[115] Wittkowski, G.: *Design and Implementation of a Workflow System in Java.* (in German) Diplomhausarbeit. Institute for Information Systems, Universität Münster 1996

[116] Wodtke D., Weissenfels J., Weikum G., Kotz Dittrich A.: *The Mentor Project: Steps Towards Enterprise-Wide Workflow Management.* Proc. 12th IEEE International Conference on Data Engineering (1996), pp 556–565

[117] Wodtke, D. *Modeling and Architecture of Distributed Workflow Management Systems. (in German)* Ph.D. Thesis, Universität des Saarlandes 1996

[118] Workflow Management Coalition. *Workflow Handbook 1997*. New York: John Wiley in association with Workflow Management Coalition 1996