A Computational Framework for Designing Interleaved Workflow and Groupware Tasks in Organizational Processes

by

Amit Vijay Deokar

Copyright © Amit Vijay Deokar 2006

A Dissertation Submitted to the Faculty of the

COMMITTEE ON BUSINESS ADMINISTRATION

In Partial Fulfillment of the Requirements
For the Degree of

Doctor of Philosophy
With a Major in Management

In the Graduate College

The University of Arizona

2 0 0 6

The University of Arizona
Graduate College

As members of the Dissertation Committee, we certify that we have read the dissertation

prepared by Amit Vijay Deokar

entitled A Computational Framework for Designing Interleaved Workflow and Groupware Tasks in Organizational Processes

and recommend that it be accepted as fulfilling the dissertation requirement for the

Degree of Doctor of Philosophy

_____ Date: July 18, 2006
Jay F. Nunamaker, Jr.

_____ Date: July 18, 2006
Mohan Tanniru

_____ Date: July 18, 2006
Daniel Zeng

_____ Date: July 18, 2006
Madhusudan Therani

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____ July 18, 2006
Dissertation Director: Jay F. Nunamaker, Jr. Date

Statement by Author

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

Signed: _____

Amit Vijay Deokar

Acknowledgements

I would like to express deep gratitude to my advisor, Dr. Jay F. Nunamaker, Jr., for his mentoring and support during my tenure as a doctoral student. Without his encouragement, advice, and continuous support during the past four years, I would not have been able to complete this dissertation. I have learnt a lot from his insights, and experiences. He has had a significant impact on my way of approaching research, teaching, and thinking at large.

I would also like to thank my other dissertation committee members: Dr. Mohan Tanniru, Dr. Madhusudan Therani, and Dr. Daniel Zeng. Dr. Tanniru was supportive of my ideas and provided me with research opportunities in industry that gave me the right motivation for pursuing this research. Dr. Madhu has provided me with detailed guidance at every stage of my dissertation work. He was always available when needed. I especially enjoyed enlightening intellectual arguments with him. Dr. Daniel Zeng provided me with valuable suggestions and feedback at various stages, which was instrumental in this work. He has been a role model of hard work for me.

I also thank the staff, researchers, and fellow Ph.D. students at the Center for the Management of Information (CMI) at the University of Arizona for their assistance in the process. Dr. Bob Briggs provided significant direction in the early stages of this research. Betty Albert made me feel at home at CMI with her helpful and jovial nature. I am also grateful for the support of the faculty and fellow students of the MIS department at the University of Arizona.

I would also like to acknowledge the following people who have been instrumental in my accomplishments. My parents, Aai and Baba, and my dear brother, Ajit Deokar, were always there for me. Without their love, patience, and encouragement, I would not have been able to accomplish this. My wife, Anuja, made this entire experience enjoyable and worthwhile, with her love and consistent support throughout the process. Thanks to her for putting up with my endless time away from home at the time when she needed my support most. My beautiful baby girl Aditi has been so wonderful to come home after long hours at school. Seeing her calling "baba. . .baba" has lifted my spirits and enthusiasm. I would also like to thank all other family members and friends for their encouragement and assistance throughout my academic career.

Dedication

To my beloved parents,

***Aai* (Mrs. Veena Vijay Deokar)** and ***Baba* (Mr. Vijay Ganesh Deokar)**

I would like to express by deepest regards to them for filling me with never-ending thirst for knowledge, providing love, patience and support throughout my education, putting in years of hard work and sacrifice, and enabling me to be where I am now.

Table of Contents

TABLE OF CONTENTS—*Continued*

TABLE OF CONTENTS—*Continued*

List of Tables

List of Figures

List of Figures—*Continued*

Abstract

Most organizations have traditionally been organized by function, and most coordination is intrafunctional rather than interfunctional. However, many organizations are finding that they must also manage processes – such as order fulfillment, new product development, and interorganizational supply chain management – that span their separate functional units and that integrate their activities with those of other organizations. These processes are essential to the well-being of organizations in a dynamic competitive environment.

In response to this, organizations are deploying large-scale enterprise information systems in order to support operational, tactical, and strategic decision making, along with information management. However, deployment of such information systems has not realized the requisite benefits due to issues such as lack of interoperability among applications due to technological evolution, constant changes to the business processes, evolving organizational structures, inherent complexity in management of distributed knowledge and resources.

To ameliorate such issues, a recent technological trend is the adoption of support tools such as Workflow Management Systems (WFMS) and groupware to support coordination between individual and group knowledge worker activities respectively. While WFMSs mostly deal with tasks involving very structured information, groupware tools deal with tasks involving unstructured information. Due to these differences, such tools are used in a fragmented manner, causing information loss. The overall guiding design principles that can be used by such process support systems are minimal, resulting in costly overheads for organizations.

This dissertation deals with the problems highlighted above from a organizational process design standpoint. The goal of the dissertation is to provide process designers with guidelines and tools that can assist them in modeling flexible and adaptable

processes. The following two research questions are central to the work described in this dissertation: (1) How can organizational processes be designed to be flexible and adaptable in dynamic environments? (2) How can collaborative activities be designed to facilitate integration with individual activities in organizational processes?

In this regard, this dissertation reports on the development of a conceptual framework to support design of organizational processes considering both individual and collaboration tasks in a unified manner. A business process is modeled as a problem solving mechanism consisting of a series of steps (also termed as process model, process definition or plan), each of which may be an individual or group activity. The task of designing business processes is considered as the development of an effective plan to solve a business process problem by searching the design space.We employ declarative formalisms from recent advances in Artificial Intelligence (AI) planning to support the task of process design. Similarly, we build on research in the field of Collaboration Engineering (CE), to propose an approach for collaborative task design. The feasibility and benefits of the approach are evaluated by prototyping intelligent build time tools for process design, and utilizing the same in the design of processes such as loan processing, and new drug discovery.

Chapter 1
Introduction

1.1 Purpose and Importance of this Research

Most organizations have traditionally been organized by function, such as purchasing, manufacturing, marketing, engineering, and accounting. This practice continues: human, physical, and financial resources are often managed by function, and most coordination is intrafunctional rather than interfunctional. However, many organizations are finding that they must also manage processes – such as order fulfillment, new product development, and interorganizational supply chain management – that span their separate functional units and that integrate their activities with those of other organizations [20]. These processes are essential to the well-being of organizations in a dynamic competitive environment.

In response to this, organizations are deploying large-scale enterprise information systems in order to support operational, tactical, and strategic decision making, along with information management. However, deployment of such information systems has not realized the requisite benefits due to issues such as lack of interoperability among applications due to technological evolution, constant changes to the business processes, evolving organizational structures, inherent complexity in management of distributed knowledge and resources.

To ameliorate such issues, a recent technological trend is the adoption of support tools such as Workflow Management Systems (WFMS) and groupware to support coordination between individual and group knowledge worker activities respectively. WFMS are aimed to facilitate coordination and integration of manual and automated processes into a cohesive whole. Groupware is aimed at assisting collaborative tasks to overcome the challenges of teamwork and inefficient meetings. While WFMSs mostly

deal with tasks involving very structured information, groupware tools deal with tasks involving unstructured information. Due to this specificity barrier, these tools are used in a fragmented manner, causing information loss [46]. The overall guiding design principles that can be used by such process support systems are minimal, resulting in costly overheads and software maintenance for organizations.

This dissertation deals with the problems highlighted above from a organizational process design standpoint. The goal of the dissertation is to provide process designers with guidelines and tools that can assist them in modeling flexible and adaptable processes. The following two *research questions* are central to the work described in this dissertation:

1. *How can organizational processes be designed with a computational representation that will allow them to be flexible and adaptable in dynamic environments?*

2. *How can collaborative activities be designed with a computational representation that will allow them to have predictable, and repeatable results as well as facilitate integration with individual activities in organizational processes?*

1.2 Process Modeling and Design

Process modeling, and software tools for the same have been studied along various lines of research including Software Engineering, Management Information Systems (MIS), Workflow Management, Artificial Intelligence (AI) and Systems Engineering (shown in Table 1.1).

There is a large body of constructive work that exists in each of these fields, that needs to be considered in a consistent manner. For example, software engineering when dealing with WFMS, works primarily under the assumption that the underlying process is stable and fixed. The focus hence is shifted on the implementation side, resulting in loss of focus on the problem of how to explore the design space. In

TABLE 1.1. Different perspectives on process design and modeling

Software Engineering	**MIS and Workflow**	**AI and Systems Engineering**
• Implementationally oriented (e.g., advent of BPEL4WS) • Process assumed to be stable or fixed • Requirements completeness assumed	• Systems analysis and design • Data flow modeling • Combined sequencing and scheduling	• Generic tools and techniques available • Business process modeling not directly addressed • Only recent interest in web service composition problems

case of MIS and workflow community, the problem of process modeling is addressed with respect to sequencing of activities and scheduling of resources. While these problems are of significant importance, they are often addressed concurrently, leading to confounded interpretation of their semantics. On the other hand, AI and systems engineering community have developed useful tools and techniques over the past few decades. However, their applications to business process modeling have only recently been studied in problems relevant to web service composition. Rather than reinventing the wheel, in this work, we have chosen to build and extend the important elements of research from these bodies of work.

1.3 Research Focus

In this regard, this dissertation reports on the development of a conceptual framework to support design of organizational processes considering both individual and collaboration tasks in a unified manner. A business process is modeled as a problem solving mechanism consisting of a series of steps (also termed as process model, process definition or plan), each of which may be an individual or group activity. The task of designing business processes is considered as the development of an effective plan to solve a business process problem by searching the design space [177]. We employ declarative formalisms from recent advances in Artificial Intelligence (AI) planning

to support the task of process design [331, 140]. Similarly, we build on research in the field of Collaboration Engineering (CE), to propose an approach for collaborative task design [58].

The research discussed in this dissertation has focused on two main concerns within the overall context of business process modeling and management (see Figure 1.1):

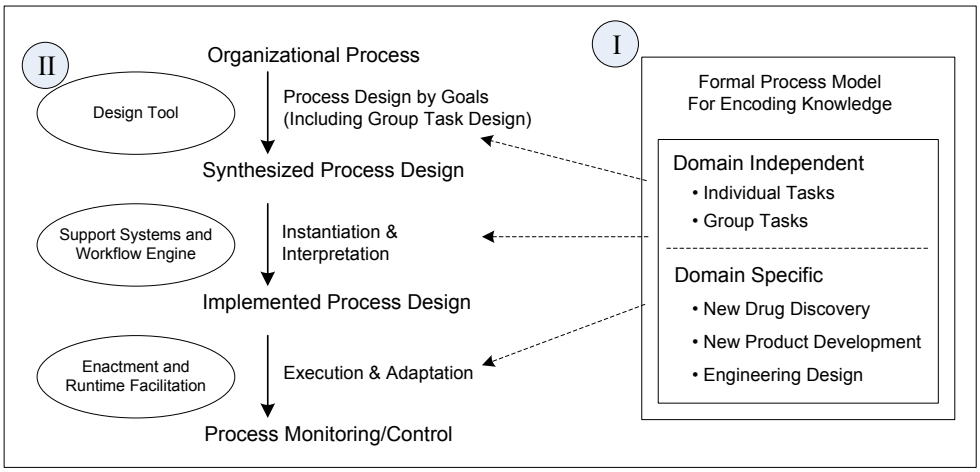


FIGURE 1.1. Research focus in the overall context of business process management

1. The first one, shown on the right side in Figure 1.1, involves proposing a formal model using declarative representation for encoding domain independent and domain specific knowledge components of business processes. The domain independent components lend itself to well studied problem solving techniques and algorithms, while the domain dependent components lend itself to specific nature of the problem in particular domains.

2. The second one, shown on the left side in Figure 1.1, involves proposing a structured process design cycle based on this formal model. The model would thus be able to support goal-driven process design. The model can also be used to support the instantiation of the design in existing process support systems

(such as GroupSystems and JBPM). Similarly, the model can be used to support the enactment and runtime facilitation of the actual processes during execution and adaptation.

1.4 Related Research Questions

As mentioned in Section 1.1, this research addresses two main research questions. The first one deals with designing processes in flexible and dynamic environments, while the second one deals with designing collaborative tasks for integrating them in workflow management systems (focused primarily on individual tasks). Table 1.2 shows related listing of research questions that are asked throughout this dissertation.

TABLE 1.2. Related research questions

Related Research Questions
• What is meant by process coordination? • What role does information technology play in supporting process coordination? • How is organizational process design related to process coordination? • What are current approaches to support organizational process modeling/design? • How do they support collaborative tasks? What are their pros and cons? • What requirements can be noted for a proposed approach based on their limitations? • How can these requirements be conceptualized through a unified approach? • What new approach can be proposed to realize the requirements? • Why are logic-based approaches a suitable candidate for process design? • What practical limitations do they have? • How are AI planning approaches different from logic-based approaches? • Why are they better suited for designing organizational processes? • How can these approaches be embedded in a system to support process design? • What are their implications for addressing the requirements noted earlier? • Why is it necessary to explicitly design collaborative tasks? • How can process structure & support for these tasks help overall process coordination? • Why is necessary to use process patterns in designing collaborative tasks? • How should the collaboration process patterns be structured? • How can the design of group task be assisted with process design support tool?

1.5 Chapter Overview

Flexible and adaptable design of organizational processes is a complex problem. In order to situate the work in this dissertation with the current body of knowledge, Chapter 2 reviews the literature on organizational systems theory, process orientation and management trends, process coordination from multiple perspectives, process support tools such as WFMS and groupware, existing process modeling approaches and their limitations. Chapter 3 presents the research methodology approaches adopted in this research, namely systems development methodology and design science. Next, chapter 4 presents the overall conceptual framework for designing organizational processes to address the research questions. Later, chapter 5 addresses computational mechanisms to realize some of the steps identified in the overall framework. In continuation, chapter 6 addresses possible mechanisms to design collaboration tasks. Finally, chapter 7 summarizes the work, details the contributions, and presents concluding remarks.

Chapter 2

Organizational Process Design and Modeling, and Process Support Systems

The study of organizational process design from an integrated perspective involves issues grounded in organizational theory, coordination science, process management, and collaboration engineering. Several process support tools have been developed over the past few decades that address the issue of process design from varying perspectives. Several modeling formalisms and standards have also been proposed to deal with different aspects of business process modeling. The following sections are a summary of the major lines of research in this area.

2.1 Organizations: A System Theoretic View

What is an organization? In the broadest sense, a business organization engages in providing goods or services. In order to achieve this, the logistic processes for the sourcing, manufacturing, storage, and delivery of these goods and services have to be performed. Input factors such as other goods and services are consumed or transformed in these processes. The logistic processes are tied with financial processes which involve raising, appropriation, and investment of funds. Both the flow of goods/services and the flow of monetary assets form the operative system of the firm. The goal of a firm's operative processes is thus the creation or use of tangible and intangible goods. Since the rationale behind any organization is to utilize the expertise of different organizational members to achieve the organization's goals and mission, the tasks performed in the operative system and the decision making about aspects of the operative system are performed by different parties involving individuals or teams. Therefore, the activities of these parties need to be coordinated in order

to contribute to the overall (primary) goals of the enterprise [422].

The system theoretic view of organizations offers a framework for the analysis of organizations, in whole or in part. Organizational process modeling, considered in this dissertation, is based on the concepts found within system theory. The following subsections provide an overview of the general principles of system theory, and outline, how these concepts relate to structuring of organizations. Parts of this discussion are adopted from zur Muehlen's extensive discussion provided in [422].

2.1.1 Systems and System Theory

General system theory deals with the description of the structure and behavior of (complex) systems [350]. It is composed of a number of independent scientific movements that originated in the 1940s. Biological system theory, regarded by many as the origin of system theory in general, was founded by Ludwig von Bertalanffy [388]. Norbert Weiner used system thinking for the analysis of the information flow within and between systems, and founded the discipline of cybernetics [404]. Simultaneously with Weiner, Claude Shannon researched the mathematical foundations of reliable transmission of messages within information systems [344]. His information theory is another example of a system theoretic movement.

A *system* in its most general form is a set of entities and relationships between these entities [5]. Formally, a system S consists of a triple (E, R, W), where E contains the entities of the system, and R contains the relationships between those entities. Systems can be nested, forming a hierarchy of systems and sub-systems. From the perspective of a lower-level system, the higher-level systems form the environment. In this case, the subordinate systems are called sub-systems, the root system is called super-system. W is the world external to the system, also called the environment. E is also called the universe of S, while R is the structure of S. E and R are non-empty sets, while the intersection of E and W is empty.

Systems can be classified according to different attributes, such as openness, complexity, and dynamics [5]. A system is said to be an *open* system if relationships exist between a system element e and an element in the environment w. On the other hand, in case of a closed system, relationships exist between the elements of the system itself. The *structural* complexity of a system describes the variety of elements and relationships within the system (i.e., their structural differences) [27]. The total number of (potentially similar) elements and relationships within a system determines the *organizational* complexity of the system [27].

The *state* of a system is the set of relevant system properties at a given point in time. The rate of change of a system's state determines the dynamics of the system [27, 5]. Accordingly, a *dynamic* system has an inner state that can be changed through inputs and that leads to an output of some sort, provided certain constraints on the system properties are satisfied. On the contrary, a *static* system does not exhibit state changes.

The representation of a system be organized by any of the following different system views: (a) *hierarchical system view*, which focuses on the composition of (super-)systems to other (sub-)systems; (b) *functional system view*, which focuses on the dynamic behavior (stateful or stateless) of a system; and (c) *structural system view*, which focuses on the organization of entities and relationships within a system.

System theoretic analysis of organization poses certain inherent dangers, as pointed out by Horváth, summarized in [422].

- Firstly, organizations can be perceived and structured as systems, but they are not systems per se. A system view of the enterprise is only a mental framework for further analysis.

- Secondly, the assumption of technological behavior is problematic, i.e., the explanation of an element's behavior through the behavior of the surrounding system. This is especially true when the entities of the system exhibit a certain

behavior that cannot be attributed to system itself. From a system theoretic perspective, an organization is comprised of roles, who exhibit more or less rational behavior, but a generalization of this behavior to the level of the organization is problematic [223].

Thus, these considerations must be borne in mind, while applying system theoretic principles for organizational process design. One crucial requirement for systems engineering efforts that can be derived from this insight is the incorporation of well-defined quality guidelines for evaluation of organizational process models. We refer back to this point in Chapters 5 and 6, during the evaluation of proposed process modeling formalisms.

Advantages of the system theoretic view are mainly tied to the reduced complexity of the studies organization through systems engineering [350].

- Complex organizational relationships can be analyzed more easily. Within the realm of organizational process design, this means that planning and execution activities, information entities, and their infrastructure can be segmented and analyzed modularly and separately, not having to consider unrelated aspects of organizational issues.

- Systems building (in this context support tools for process design) is helpful for the isolation of interesting (organizational) system dimensions. Through the exclusion of irrelevant details, the organizational complexity of the resulting system is reduced.

- A system view of the organization is a valuable instrument for the design of technical systems that are employed within an organization, such as the information technology infrastructure involving decision support systems and collaboration systems.

- The view of the organization as a system fosters the analysis of ongoing system changes, thus enabling the support for adaptive and dynamic business processes with control and regulation of system aspects.

2.1.2 Organizations as Socio-Technical Systems

In light of the system theory discussed above, organizations can be perceived as socio-technical systems that are goal-oriented, have a specific purpose, have relationships to outside entities (i.e., they are open systems [178]), exhibit a significant complexity both in terms of structure and organization over time (i.e., they are dynamic systems [322]). The constituting characteristic of organizations as a system is the (artificial) demarcation between the organization and the outside world. Typically this demarcation is based on the specific goals and objectives of the organization. This perspective also aligns well with the process orientation observed in today's organizations, as discussed in Section 2.2. This viewpoint of organizations is assumed through this work.

In the same vein as the system theoretic viewpoint, Weber considers an organization to be a set of constraints on the activities performed by roles/agents. This perspective considers the process of bureaucratization as a shift from management based on self-interest and personalities to one based on rules and procedures [391].

2.2 Process Orientation in Organizations

Modern organizations are observed to focus on processes with their core competency as well those supporting them directly or indirectly. A brief discussion on this evolution is given below.

Much of 20^{th} century organizational research is built on the pioneering works industrial engineers such as Fayol [122] and Taylor [366]. One one hand, Fayol's research focused on the managerial structure of an enterprise, while on the other

hand, Taylor's research focused on operational enactment of tasks as well as design of organizational structures that support the efficient execution of these tasks. Taylor's ideas have dominated organizational research until the 1980s and can still be found in many organizations today. Until the 1980s, the functional separation of tasks was appropriate for the then industry environment. Since then, the industry environment has changed considerably over the past decades, increasing market segmentation as well as shorter product life-cycles, amongst other factors. This has led enterprises and researchers to investigate organizational structures better suited to adapt to changing market conditions, product portfolios, and enterprise infrastructures. In this regard, business processes have become a focal point of organizational research.

The alignment of organizational structures along their business processes has been discussed in the organizational literature as early as 1930s. Most notably, Chapple and Sayles are among the early proponents of process-orientation for organizational structures [67]. As mentioned above, despite these early efforts, a functional separation of tasks, and the resulting functional or divisional structures dominated the corporate practice until the 1980s, when changing market conditions and increasing competition led companies to investigate the efficiency of their process structures.

With this instigation, process thinking attracted more interest from both researchers and practitioners. Numerous process related management practices emerged in the 1990s, following the Total Quality Management movement, leading to a resurged interest in process concepts and their implementation. Most notable among them were Business Process Improvement [170], Business Process Innovation [77], and Business Process Reengineering [163, 162]. This led practitioners to believe that process-centered view of organizational design can yield dramatic improvements in organizational performance. On the other hand, researchers came to a similar belief based on their field and case studies [134, 252]. In fact, some sociologists adopted the view that "social reality happens in sequences of actions located within constraining and enabling structures" [3]. Organization theorists also argued that process models

provide a unique perspective on innovation [376], strategic change [375], and organizational behavior in general [269]. Whether this wave of interest in processes is regarded as a case of theory leading practice, or practice leading theory [40], the value of process-oriented approach to organizational design is certainly evident.

2.3 Process Management

Managing corporate processes is a natural follow-on associated with the shift toward process-oriented organizations. Process management can be viewed as the application of the management cycle with a focus on organizational processes. The organizational literature is abundant with several taxonomies and frameworks. One of the influential categorizations of management functions was given by Gulick [160], who built upon the work of Fayol [122]. His classification consists of seven distinct functions: *planning* (at a strategic level), *organizing* (formal structure of authority for controlling), *staffing* (knowledge workers for various roles), *directing* (continual decision-making), *coordinating* (connect various parts of work process), *reporting* (continual assessment through information provisioning), and *budgeting* (creating and supervising budget for meeting financial and resource constraints). Later Koontz and O'Donnell proposed considering coordinating, reporting, and budgeting under a single function of *controlling* [214]. These management phases are not merely isolated sequences executed at the higher management level. The sheer complexity of large organizations rules out implementing these stages at a topmost level. Instead, each part or level of an organization performs, at least to some extent. Moreover, these phases occur in a cyclical manner, enabling the organization to continuously control itself, in order to maintain its viability and development.

The cyclic nature of the overall management process corresponds with the cybernetic feedback loop shown in Figure 2.1 [119, 405, 218]. The planning unit provides target values for the output of the regulated system to the control unit. Input for

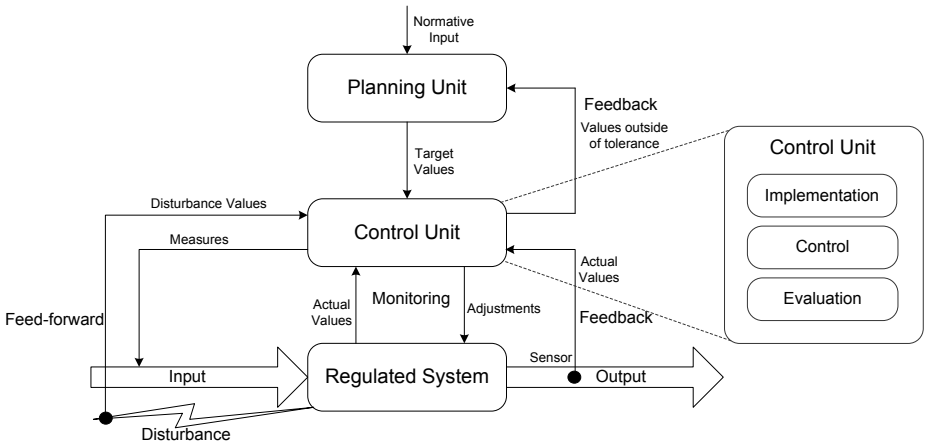


FIGURE 2.1. Cybernetic feedback loop [119, 422]

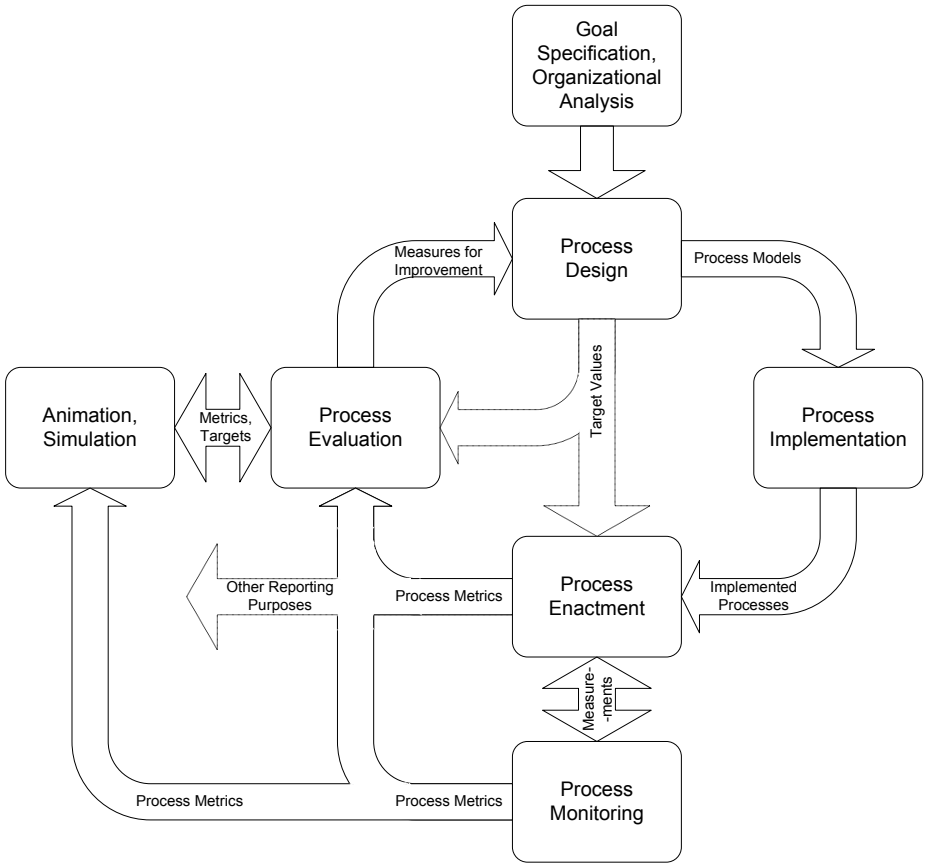


FIGURE 2.2. Process life cycle [421]

the planning unit comes from the normative management level, which determines overall strategies, rules, and guidelines for the organization. The control system has to ensure that the regulated system produces an output within the tolerances defined for the target values. This unit consists of three sub-units: evaluation, control, and implementation. Finally, if a regulated system allows a view of its internal processes, it can be monitored during the execution phase. If the system also allows a modification of its processes, real-time adjustments can be administered through the control unit.

In the same vein, a management-oriented process life cycle, proposed by Michael zur Muehlen is depicted in Figure 2.2 [422]. This model is a fine grained version of the process life cycle by Neumann, Probst, and Wernsmann [287]. Initial analysis of the process goals at a desired organizational level involves knowledge gathering (both tacit and explicit [290]) regarding the process, organizational structures and rules surrounding the system. This phase is followed by a process design phase, during which the overall process structure is engineered, the process model is designed, and the resources involved in the process execution and responsible for its results are specified. This includes the modeling of organizational structures as well as the definition of task assignment policies and conflict resolutions mechanisms. With process models as the input, the process implementation phase involves designing the infrastructure for business process support and integrating the solution with the surrounding information systems. During process enactment, individual process instances are derived from the process model and they may be coordinated by the process automation infrastructure. Concurrently, from an administrative perspective, process monitoring takes place, which involves measuring the performance of the process management system (on the technical side) as well as measures such as the length of work queues, resource idle times (on the organizational side) are supervised. Next, the process evaluation phase involves a post-execution analysis of process instances based on execution protocols (audit trail). Results from this analysis can serve as the basis for

planning of resource capacities, further leading to adjustments of the process structure, which can be pre-tested with simulation and animation for expected process performance. It can be noted this process life cycle model bases itself on a synthesized process model (workflow schema), which is designed by process designers using their experience in the domain.

The models above express valid approaches to process management. We build on the notions expressed in the above process management models and propose a structured process design cycle (SPDC) in Chapter 4. While the overall cyclic nature of process management is kept in mind, the focus of SPDC is on the organizational process design aspect, suggesting use of computational mechanisms and tools to support the design phase. It is also emphasized in SPDC that an interplay between design and execution can help a better resultant coordination mechanism. Also, a major difference as will be noted later, is the generative instance based process modeling paradigm, rather than the synthesized predefined process modeling paradigm, thus allowing for adaptability and flexibility of the processes.

2.4 Process Coordination

A critical part of process management, as pointed out above, is the coordination of different activities within the organizational processes. In a sense, the goal of process management is to ensure efficient process coordination. Thus, along with the drive toward process-centric organizations, achieving coordination within these organizational processes is becoming increasingly critical.

We all have an intuitive sense of what "coordination" means. For example, a well-organized celebration event, a smooth publishing process for an academic journal, a software created by a team of software engineers, all represent coordination in some respect for the concerned process. However, it is helpful to clarify the precise notion of "coordination" and how it relates to organizational process design. We refer to

some such efforts from the literature and situate our work in the context.

2.4.1 Three Levels of Group Work

Collaboration research community has focused on the process coordination problem from the perspective of group work. Each organizational process, from this perspective, can be considered to be a collaboration process with differing levels of group work. A classification of group work suggested by Briggs and Nunamaker is relevant to this discussion on process coordination. According to Briggs [55] and Nunamaker et al. [297], group work falls into three general categories: individual, coordinated, and concerted, as illustrated in Figure 2.3.

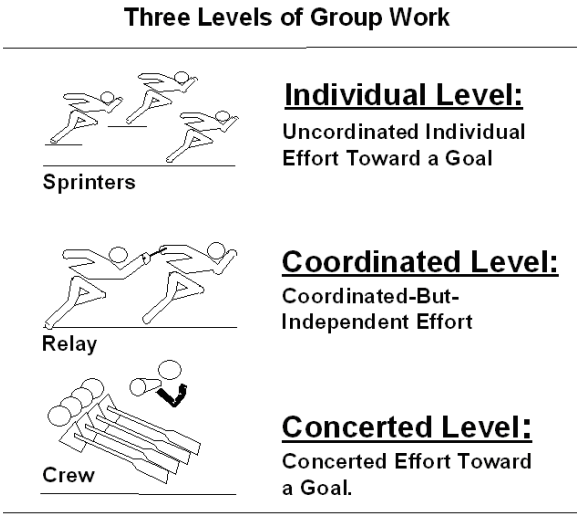


FIGURE 2.3. Three levels of group work [55, 297]

Uncoordinated : Sometimes the effort spent by a group of individuals towards a common goal is largely uncoordinated. An example of this is a manager of a business organization who encourages her employees to become more physically fit. Assume she tells employees they will be healthier and more productive. However, she does not coordinate or plan their exercise program or seek reporting on whether employees are actually exercising or not. Moreover, individual employee efforts to exercise are

independent of the work done by other employees. Typical information technology tools to support such efforts include word processing, presentation graphics packages, Decision Support Systems (DSS) such as data analysis and report generation.

Coordinated : When working in a coordinated effort, participants work individually on some aspect of the work and then hand the work off so that other individuals can continue the work. Thus, this type of work is individual but coordinated effort (ICE). With this type of collective effort, coordination decisions must be made by addressing questions regarding who will do which activity in the process and in what order. Moreover, a priori or during the process, decisions must be made about what information needs to be shared and with whom? A process of providing technical support for a product is an interesting example of an ICE. The original help desk operator may obtain enough information from a customer to be able to route the call to one of multiple internal experts who possesses the expertise and responsibility to help with that specific type of problem. The initial operator may collect and record basic information about the customer and the problem and then pass this information on to the specialist. If the problem cannot be fixed remotely by the specialist, the specialist may decide whether to ask the customer to mail the product in for repair or to recommend that the company send out a replacement product. Note that if an ICE is an important and nontrivial process, the work must be designed and coordinated to be both effective and efficient. ICE processes are typically supported by current WFMS.

Concerted : A concerted effort (CEff) is the most intensive type of collaboration. Interaction among group members is high and work is conducted synchronously to accomplish the group's goal. An example of a CEff is service department employees who meet periodically to identify ways to improve customer service. Employees may meet face-to-face, through a Group Support System (GSS), or via teleconference to generate ideas of how to improve customer service. Group memory [300] may

be provided via white board or GSS so that employees can see ideas generated by other employees. After that, group members vote to identify some of the best ideas, use group discussion to refine those ideas, and then plan the steps and timelines to implement the ideas. In this example, each phase of the work is synchronous and there is high interaction among participants.

Table 2.1 summarizes these three categories of group work in relation to the level of participant interaction and work synchronicity.

TABLE 2.1. Categories of group work

Type of Combined Effort	**Interaction Level**	**Synchronicity**
Uncoordinated Collective Effort	None or Very Little	Unnecessary
Individual but Coordinated Effort (ICE)	Moderate, Periodic	Asynchronous
Concerted Effort (CEff)	High	Synchronous

Hybrid ICE and CEff Tasks : It is common for some group tasks to interleave coordinated and concerted phases. A particular approach for the collaborative preparation of a group document [238] is an example of this. During document preparation, a group meets face-to-face or in a distributed synchronous meeting to set specific goals, develop a document outline, assign portions of the document to group members, and develop a completion timeline. Then the project moves to an ICE phase where individuals work on parts of the document and later examine and annotate parts of the document written by other participants. Next, the larger group meets to work in a concerted fashion to make or approve final changes to the document. Then the process returns to an ICE for the formatting and dissemination of the document. In such hybrid examples, the transition between concerted and coordinated work often occurs multiple times. Increased globalization, outsourcing, and the distributed nature of organizational processes make it is increasingly important to plan and coordinate well-specified and routine tasks as well as some highly unspecified and situated

tasks [46]. ICE tasks that are well-specified and routine are often supported by process support systems such as WFMS [397], while CEff tasks are often supported by GSS [297]. From an organizational perspective, many organizational processes consist of interleaved ICE and CEff tasks, so achieving efficient and effective processes is important [92, 89].

2.4.2 Coordination Theory Approach

Malone and Crowston have taken an interdisciplinary approach to researching process coordination. In their earlier attempts to characterize coordination, they proposed a "broader" definition of coordination as *the act of working together harmoniously* [248]. They analyzed this broad definition with respect to the components and the coordination processes associated with them, summarized in Table 2.2 (see [35, 36, 41, 246, 260, 267] for related decompositions of coordination). Based on this analysis, they proposed the following "narrower" definition of coordination, which captures the desired notion in the context of this work: *Coordination is managing dependencies between activities* [249]. This definition is reflective of the intuition that, if there is no interdependence, there is nothing to coordinate. Also, it is consistent with the emphasis on the importance of interdependence, as indicated by the long history of organizational theory [368, 133, 223, 307, 326, 172, 325]. This definition suggests an inclusive sense of the word *coordination*. For example, even though the words "cooperation" and "collaboration" each have their own connotations, much of what they describe falls within the above definition of coordination. It is also often possible to analyze "conflict" and "competition" from the point of view of some higher level goals and see that conflict at one level leads to coherent results at another level. For instance, different groups in a company may compete for resources and people, but this very competition may contribute to the company's overall ability to produce useful products. In summary, Malone and Crowston take the stance that even though it is

often important to distinguish between concepts like cooperation, collaboration, and competition, including them all in the same conceptual notion allows for examining their relationships and their relative advantages and disadvantages.

TABLE 2.2. Components of coordination [248, 249]

Components of Coordination	**Associated Coordinated Processes**
Goals	Identifying goals (e.g., goal selection)
Activities	Mapping goals to activities (e.g., goal decomposition)
Actors	Mapping activities to actors (e.g., task asssignment)
Interdependencies	"Managing" interdependencies (e.g., resource allocation, sequencing, and synchronizing)

Coordination theory emphasizes that understanding the nature of interdependence between activities is important for determining how organizations are structured, which can further lead to analysis and managing of organizational coordination processes. A classification of dependencies and related coordination processes proposed by Malone et al. is shown in Table 2.3 [248].

TABLE 2.3. Dependencies and coordination processes [248]

Dependency	**Description**	**Coordination Processes**
Prerequisite	An activity depends on the output of another activity	Activity ordering
Shared Resource	Multiple activities require the same resource	Resource allocation
Simultaneity	Two activities must be performed at the same time	Activity synchronization
Task/Subtask	Top-level goal is dependent on the achievement of other goals	Goal decomposition

Such a classification can help generate possible alternative ways of coordinating in a particular situation. In that, a situation may be characterized with respect to

the kinds of interdependence involved and then use a "library" of interdependencies and their associated processes as a map to generate a set of alternative processes that could be used to manage the interdependencies. This ability to characterize a space of possible coordination processes for a given set of activities can be helpful in understanding how existing as well as new process support coordination tools could lead to new ways of organizing business processes.

We illustrate with an example from Malone et al. [248] how knowing the interdependencies in a situation may suggest alternative ways to manage them. The example is based on extended field studies of engineering change processes in several manufacturing organizations [74]. In the case of design and manufacturing, design of the product serves an a common object, which facilitates important kinds of interdependencies between the associated activities. This particular example involves routing engineering change notices to concerned engineers whose work is likely to be affected by a given change, even when the person making the change does not know who else it will affect [221, 246]. Four possible ways identified for managing interdependencies are listed below [248]:

1. At a minimum, the designer must create a design and give it to the manufacturer to build. One simple effect of CAD systems, for example, is to make this transfer process easier.

2. The designer and the manufacturer can negotiate what the design should be, for example, by iterating the design process or in joint meetings. A variety of electronic meeting support and communication tools could help this process and could make it more desirable relative to alternative ways of managing the same interdependencies.

3. Sometimes the need for explicit negotiation can be eliminated by moving some of the knowledge about the constraints of either task from one engineer to another. For instance,

(a) Some of the manufacturer's knowledge (the knowledge about the manufacturing constraints, not about how to do the manufacturing) can be made available to the designer, for example, by training the designer in methodologies such as design for manufacturing or by embodying the knowledge in an intelligent CAD system.

(b) Some of the designers knowledge can be transferred to the manufacturer. For example, if a system like gIBIS [71] is used to capture more of the designers intent as well as the details of the part, the manufacturing engineer might be able to change some details of the design to make the parts easier to build while preserving the intent.

(c) A third party, such as a common superior, may be able to resolve problems as they arise or to give enough initial direction that problems do not arise.

As explained by Malone et al. [248], such analysis seems to be easily transferred to other domains. For example, a bank and a potential borrower have to agree on a common object, a loan. The typical approach seems to be case (1) above: the bank offers a loan with its standard terms and a person who wants the loan takes it or leaves it. In some cases, the bank and the borrower negotiate the details of the loan, case (2) above. Finally, one can imagine transferring some of the banks knowledge about making loans, for example, to a computer program that a potential borrower could run to explore possible loan conditions (case (3a)), or to a third party who would suggest which bank would be best for a given applicant (case (4)).

These process design notions are extremely valuable and need to be leveraged by embedding them in organizational process modeling tools. A potential way to do this is by capturing domain independent as well as domain dependent interdependencies between activities of a process in a domain description and then allowing an automated system to reason about the constraints to generate possible alternative process models. Thus, building on notions of coordination theory, this research focuses on

the above aspect by addressing the following two related research issues: (1) How can we represent organizational processes, composed of subprocesses and activities, to allow reasoning capabilities with the help of a computer-supported tool? (2) How can generative and alternative process design be supported with such a process design tool?

Processes Underlying Coordination : Another important insight provided by the coordination theory is that of understanding the nature of different coordination processes by characterizing them in terms of successively deeper levels of underlying processes, each of which depends on the levels below it [249]. Table 2.4 shows a diagram of these levels. For instance, some form of group decision-making in most of the coordination processes listed above in the last column of Table 2.2 (e.g., what goal will be selected or which actors will perform which activities). Group decisions, in turn, often require members of the group to communicate in some form about the goals to be achieved, the alternatives being considered, the evaluations of these alternatives, and the choices that are made. This communication requires that some form of "messages" be transported from senders to receivers in a language that is understandable to both. Finally, the establishment of this common language and the transportation of messages depends, ultimately, on the ability of actors to perceive common objects such as physical objects in a shared situation or shared information in a computer database (e.g., see Suchman [362]).

The order of these layers reflects the observation that each layer uses processes from the layers below it. For instance, group decision-making requires that the members be able to communicate in some way. However, unlike most network protocols, there are also times when a layer uses processes from the layers above it. For instance, a group may sometimes use decision-making processes to extend the common language it uses to communicate (e.g., see Lee and Malone [227]), or a group may use coordination processes to assign decision-making activities to actors.

TABLE 2.4. Processes underlying coordination [248, 249]

Process Level	**Components**	**Examples of Generic Processes**
Coordination	goals, activities, actors, interdependencies	identifying goals, ordering activities, assigning activities to actors, allocating resources, synchronizing activities
Group decision-making	goals, actors, alternatives, evaluations, choices	proposing alternatives, evaluating alternatives, making choices (e.g., by authority, consensus, or voting)
Communication	senders, receivers, messages, languages	establishing common languages, selecting receiver (routing), transporting message (delivering)
Perception of common objects	actors, objects	seeing same physical objects, accessing shared databases

2.4.3 Other Related Approaches to Process Coordination

The coordination science initiative by Malone and Crowston is one of the significant efforts toward conglomerating a body of principles about how activities can be coordinated. Ongoing research can be noted in many different disciplines aimed toward domain specific applications of coordination science. A comprehensive discussion of such related work can be found in [249]. We note below few of the approaches that are related to work in the area of workflow and groupware.

Holt has described a theoretical language *Diplans*, motivated by Petri nets, for designing coordination tools in the domain of distributed or parallel systems [183]. Winograd and Flores have proposed a communication-based approach – a language/action paradigm based on speech acts [342] – for workflow modeling as well as analyzing group action [408, 411, 127, 409, 410], which formed the basis of systems such as the Coordinator and Action Workflow. Malone et al. have developed a system called Information Lens for helping people share information in organizations [251, 247], founded on the ideas from organization theory about flexible organizational structures called adhocracies [267] and ideas from artificial intelligence about blackboard architectures for sharing information among program modules [289]. Con-

klin and Begeman [71] and Lee [225] have described systems to help groups of people record the structure of arguments (e.g., positions, arguments, and counterarguments), grounded in ideas from philosophy and rhetoric about the logical structure of decision-making. Gerson and Star study the problem of office systems having to deal with inconsistent knowledge bases and procedures. They take a social science approach, hinging on developing local closures to the problem, by analyzing analogous problems in human organization [139]. Marschak and Radner, take an economic standpoint in proposing a *team theory*, where they analyze how information should be exchanged when multiple actors need to make independent decisions but all performers share a common final goal [257].

2.4.4 Summary

The discussion on levels of group work (Section 2.4.1) really points to the fact that process coordination tools are found to be fragmented based on each group work level. As illustrated by above example, ICE and CEff tasks are often interleaved in real world settings, but are unfortunately supported by different coordination applications. This classification can be contrasted with the abstractions of the underlying coordination processes, presented in the coordination theory discussion (Section 2.4.2). Processes with concerted efforts in Table 2.1 align with the group decision-making tasks in Table 2.4, enabled by groupware technologies such as GSS. Also, ICE in Table 2.1 align with the coordination tasks in Table 2.4, typically enabled by workflow technologies. Communication and perception of common objects is not explicated in consideration of group work levels. However, it is noted that research in Computer Supported Cooperative Work (CSCW) has typically focused on these (communication and perception of common objects) aspects, whereas GSS has focused on the group decision-making aspects [91]. This calls for a unified approach to effectively achieve information flow in organizational settings by integrating available process

support tools. The SPDC framework presented in Chapter 4 considers this aspect of coordination, aimed at bridging the gap between structured process coordination tools (WFMS) and unstructured process coordination tools (groupware).

Next, we discuss the role of information technology in supporting process-oriented organizations, beginning with a brief history of process support technology.

2.5 Process Support Systems: Historic Perspective

The need for capturing and supporting organizational processes has led researchers in several disciplines to investigate the issues of understanding, modeling, analyzing, and building processes and to support them through coordination and collaboration of humans and information systems.

The idea of software support for corporate processes has been researched since the late 1960s. At the time – the late 1960s – application development was driven by increasing volumes of data, which were of interest to different recipients inside the organization. However, real-time access to computing technology was not economically feasible, even for large corporations. The idea of time sharing computers, which allowed end user to manipulate the data stored in the system, was only addressed by researchers like Douglas Engelbart at the Stanford Research Institute (SRI), who demonstrated the prototype of an office system called NLS (oNLine System) in 1968 [130]. The first commercial efforts in the area of real-time applications was the Semi-Automatic Business Research Environment (SABRE) by IBM and American Airlines, a travel reservation system that went into production in the 1960s and commercially operationalized in 1976 [21].

Jablonski and Bussler identify seven fields as the conceptual ancestors of what is today known as workflow management technology, namely, office automation, database management, e-mail, document management, software process management, business process modeling, and enterprise modeling and architecture [187]. In addi-

tion to these fields, Sheth mentions distributed object management, imaging technology, transaction processing monitors, workgroup software, and Internet technology as domains that are influential to workflow management technology [346].

2.5.1 Office Automation Technology

Figures 2.4 and 2.5, adapted from Muehlen et al. [422], gives an overview of historical development of office information systems and how it led to the development of workflow technology. Despite claims that the development of workflow applications is tightly interwoven with the business process reengineering movements of the early 1990s [129], the origins of workflow technology can be traced back to the late 1970s. Research in office automation technology flourished between 1975 and 1985, studying the application of information technology to office environments, and laying the groundwork for the development of industrial workflow applications through the analysis of technology support for administrative processes [245, 302]. Increased computing power and higher accessibility of information technology were important factors in fueling this stream of research [60, 401]. As stated by Ellis and Nutt, the focus of office automation technology was "to reduce the complexity of the user's interface to the (office information) system, control the flow of information, and enhance overall efficiency of the office" [110].

Ellis and Nutt were instrumental in the early advancement of the area of office information systems, primarily during their research at Xerox PARC in late 1970s. They developed several generations of such systems, called Officetalk-Zero [110], Backtalk (testbed for Officetalk-Zero) [303], Officetalk-P [107], and Officetalk-D [108]. On the design side, these systems represented business processes as Information Control Nets (ICN) [107], a variant of Petri nets. The idea was to structure business processes using a form-based analogy for graphical user interfaces, allowing interaction between knowledge workers.

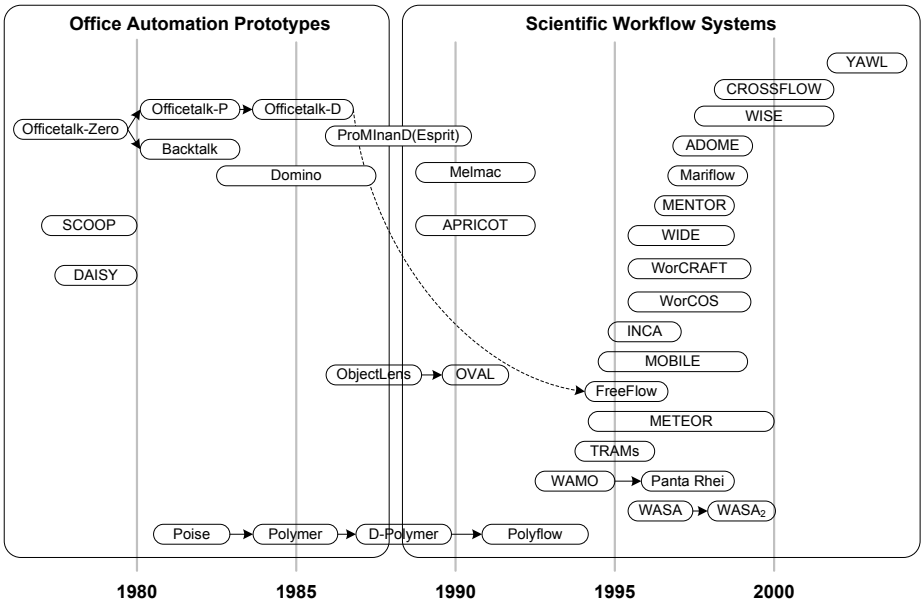


FIGURE 2.4. History of WFMS (part A) (adapted from [422])

Similarly, Zisman developed an office automation system called SCOOP that used Petri nets to represent business processes, with augmentation supporting multiple trigerring of activites [420]. Later, in 1983, Kreifelts and his colleagues initiated the development of the DOMINO system [216], which was later used by Olivetti as the basis of the commercial X_Workflow system [412]. While most of the contemporary systems used Petri net-based models for representation, Hammer and his colleagues at IBM developed a Business Definition Language (BDL), as a high level programming language for business processes [164]. BDL was intended to create formal office processes on the fly, but had limited commercial success.

As observed in the literature, research interest in office automation ceased by mid-80s [364]. However, two research developments spun off that targeted beyond the boundaries of traditional office automation: Computer Supported Cooperative Work (CSCW)/Groupware, and Workflow Management. Each of these developments are discussed in later sections. But we first review other related technologies that contributed to the development of workflow management technology.

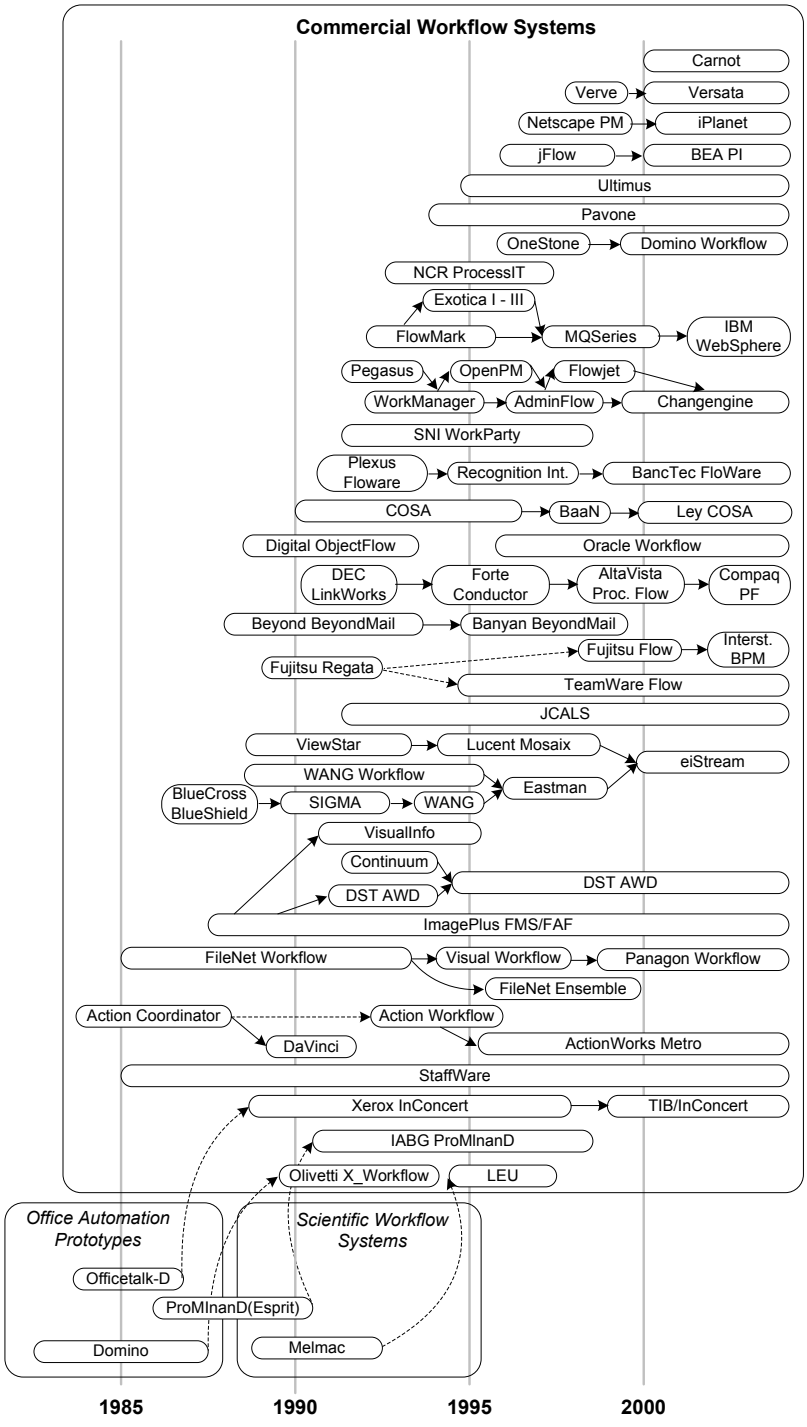


FIGURE 2.5. History of WFMS (part B) (adapted from [422])

2.5.2 Other Related Technologies

Along with office automation technology, roots of workflow management technology can be found in the following three primary areas of research [422]:

Document Management Systems : Document management systems have given rise many document-oriented WFMS in which the focus is the process object (in most cases an electronic document) [138]. This view has an impact on the process modeling paradigm employed by these systems. While activity-based modeling methods [333] represent a process as a sequence of tasks with associated performing roles, document-oriented workflow applications are often based on state-based modeling methods. These describe a process in the form of the legal state-changes of the process object, e.g. a document that may change from drafted to reviewed to either rejected or approved [363]. FileNet's Business Process Manager is an example of such a workflow project based on a combined workflow and document management infrastructure [124].

E-mail Applications : Another predecessor of workflow technology are advanced e-mail applications. Workflow management technology extends the functionality of standard e-mail system (simple point-to-point routing of messages with one or more recipients) to include a list of recipients that are addressed sequentially (e.g. to coordinate the editing of a document) [419]. As opposed to activity-oriented or process-object-oriented perspectives, messaging-based workflow systems present interesting aspects such as evolutionary workflow modeling or the support of unstructured processes where the actual sequence of activities is determined at run time. Another notion supported by this modeling paradigm is that of a network of processing workstations that are supplied with process objects in a structured way [39].

Database Management Systems : WFMS rely on database technology to store business process models, the current state of workflow instances, and information elements

relevant to the workflow instance execution [418]. Two areas of database management are particularly relevant to workflow technology, namely, transaction management, and active rules. From a transaction management perspective, a workflow instance can be perceived as a long-running transaction with multiple sub-transactions (i.e. workflow activities). Analysis of transaction concepts to ensure the integrity of the workflows during execution, without affecting the overall efficiency of the organizational process is one of the research issues in this area. From the perspective of active database research, WFMS can be implemented through triggers and active rules in databases, which monitor system conditions and raise triggers upon the detection of specified conditions. The sequence of activities would thus be implemented as a collection of database actions [234].

2.6 Workflow Management Technology: WFMS and BPMS

2.6.1 Commercial Use of Workflow Technology

Mid-1980s saw a pause in the research in office automation and the beginning of commercial exploitation of workflow technology as shown in Figures 2.4 and 2.5. The development of these first generation of workflow systems was built on office automation technology and fostered with other related technologies such as database, imaging and document management technology, and e-mail applications, as discussed above.

WFMSs are a new breed of information technology aimed to facilitate automation of business processes by coordination and controlling the flow of work and information between participants [360]. They can be also viewed as a form of middleware or glue to facilitate integration between back-end information systems, including legacy applications, into an enterprise-wide system. From a coordination science point of view, they function as an enabling technology for achieving coordination within and between processes for streamlining and automating routine manual business processes,

and integrating both manual and automated processes into a cohesive whole. WFMS are now used not only for coordinating office tasks, but also for managing inter-organizational information flows. In essence, workflow technology is aimed at leveraging the value of existing information system infrastructures and helping enterprises in the transition toward a process-oriented organization.

Today, the commercial workflow market is extremely fragmented in terms of process modeling formalisms, features, and breadth of application. Many stand-alone systems (WFMSs) have appeared in the workflow market (e.g., IBM's Websphere MQ, Ultimus, FileNet's Business Process Manager) over the past few years [423]. Noticing the value for workflow management, many companies are furnishing their existing software packages by embedding one or more workflow management features, as can be seen in the case of Enterprise Resource Planning (ERP) systems (e.g., SAP R/3) and e-business Enterprise Application Integration (EAI) infrastructures (e.g., Vitria, BEA WebLogic Integrator, and TIBCO) [423]. Currently, WFMS capture, automate, and monitor predefined/static processes that consist of prescribed activities, and provide tools for assigning activities to people and external applications. WFMSs have been implemented in several domains ranging from insurance claim processing and loan approval to Operations Support Systems (OSS) in telecommunications, providing process-based coordination and application integration. The current state of the workflow market versus the findings of researchers in the workflow domain are discussed by Abott and Sarin [4], Georgakopoulos et al. [138], Alonso et al. [16, 14], and Du et al. [101]. The underlying technology and applications of workflow management systems are well described in [234, 386] and an overview of research issues from various perspectives is provided in [45, 360].

It has been seen that successful WfMS implementations can result in significant process cycle time reductions, cost reductions, improved accuracy, greater coordination and control over processes, and greater knowledge worker satisfaction [360]. For example, as indicated by Ader [6] (reported in [360]), workflow management can re-

sult in productivity gains of 5% to 30% and cycle time reductions of 30% to 80%. However, it is to be noted that like any other information system, issues related to technology adoption such as poor change management, resistance from rigid bureaucratic organizations, and lack of sustained top management support are points of concern, leading to unsuccessful process management projects with WFMSs [152].

2.6.2 Different Types of Workflow

Workflow can classified in several different ways. The most widely accepted classification, one that has been endorsed by WfMC, divides workflow in four categories: production, administrative, ad hoc, and collaborative [12, 379, 65] (refer Figure 2.6).

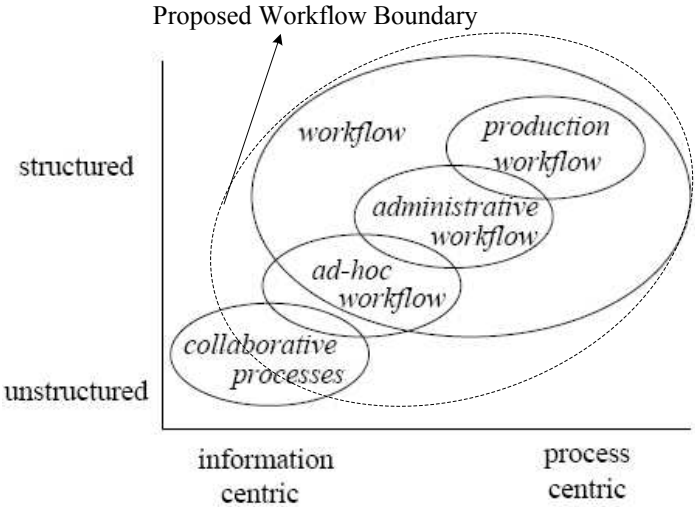


FIGURE 2.6. Workflow classification (with proposed boundary) (adapted from [379])

Organizational processes that can benefit from the structure and coordination with the help of IT support systems such as WFMS are expected to possess the following characteristics [89]: (1) goal-driven (with explicitly defined outcomes), (2) high value for the organization, and (3) needs of process coordination and information processing. Figure 2.6 shows the spectrum of business processes. Production workflow is concerned with processes involving highly structured and repetitive tasks with

almost no variations [234] (roots in document imaging systems) (e.g., loan processing). Administrative workflow concerns itself with processes with defined procedures (structure), although each workflow instance or case can have different work performed (roots in office automation). Thus, alternative routing of a case is possible, but needs to be predefined (e.g., process of an expense report approval). Ad hoc workflow relates to processes where the procedure is not defined (completely) in advance and is individualized on a per instance basis. The workflow model is not predefined for the business process that it is serving (e.g., checking customer credit) [389]. Collaborative workflow involves teams or people working together on activities where input from several different roles and members is needed (e.g., strategic planning, design review).

Most of the research and development in workflow has focused on production, administrative, and to some extent ad hoc workflows (as shown in Figure 2.6). One of the goals of this research work is to broaden these boundaries to encompass ad hoc and collaborative workflows. We concern ourself to the modeling and design issues in this work. The main research issue to be addressed to support ad hoc workflows is adaptability and flexibility. Also, to support collaborative processes from within WFMS, it is desired to bring process structure and support to those processes [300]. We've focused our research efforts in this direction.

2.6.3 Standardization in the Workflow Area

As indicated above, there are differences in how each workflow vendor perceives workflow management to be, which eventually reflects in end products, i.e., the WFMSs that are available for organizations to deploy.

Due to the interdisciplinary nature of workflow research, the underlying concepts and terminology is varied in the literature as well as WFMSs. Current efforts are attempting to get researchers and vendors to converge on a common foundation. From

an embedded WFMS point of view, standardization of integration interfaces can allow reuse of workflow components and reduction in implementation times. Similarly, in case of inter-organizational workflows (B2B processes), standardization allows for plug-and-play type solutions, reducing the risk for companies to lock into proprietary solutions. Several efforts are underway to achieve such standardization.

The Workflow Management Coalition (WfMC), formed in August 1993, is presently one of the most significant of the efforts attempting to standardize workflow system interfaces. The WfMC standards are aimed at providing recommendations for workflow vendors, although they are not obligated to implement them. The Terminology & Glossary work of WfMC is aimed at standardizing the use of terminology across the industry [398].

According to WfMC, a *business process* is "a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships" [398].

Based on this notion of business process, a *workflow* can be defined as a specific representation of a process, whose formal coordination mechanisms between activities, applications, and process participants can be controlled by an information system, referred to as the workflow management system [422].

A *workflow management system* is defined by the WfMC as "A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications" [398].

Related to this definition is the consideration of a process definition or a workflow model. According to WfMC, a *workflow model* is "the representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consists

of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc. [398]"

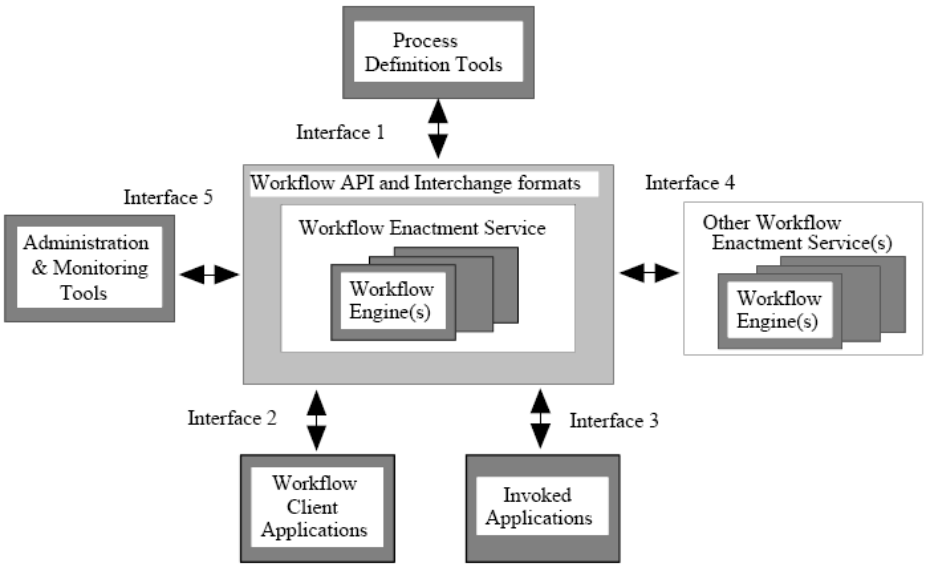


FIGURE 2.7. Workflow management coalition reference model [398]

In addition to the WfMC Terminology & Glossary, WfMC's main contribution is the WfMC Reference Model and a set of interfaces, called Workflow API and Interchange (WAPI) based on that model (shown in Figure 2.7). This is an attempt at standardizing the architectural elements of workflow systems and the interactions between those elements. At the center of the model is a Workflow Enactment Service (WES), comprised of one or more workflow engines. A WES provides services through the WAPIs to external workflow-related tools. These include Process Definition Tools for defining processes, Workflow Client Applications for handling user requests for work, Third-party Applications that need to communicate data and operations to the WES, other WESs for providing interoperability between enactment services, and Administration and Monitoring Tools for data gathering for process improvement activities.

While WfMC's effort for standardization is noteworthy, several points can be

critiqued upon. From a process modeling standpoint, the WfMC Interface 1 provides a generic process description format, namely, the Workflow Process Definition Language (WPDL) [399]. WPDL is aimed at the exchange of workflow specifications between different business process modeling tools and WFMSs. WPDL itself is independent of specific system implementation. However due to conflicting modeling concepts used in the workflow community by researchers and the industry, the convergence on WPDL turns out to be difficult. As an example, Petri net based tools such as COSA [379] are very different from speech-act based tools such as Action Workflow [261].

In 2004, an XML version of WPDL, called XPDL was published by WfMC [400]. A number of open-source projects such as jBPM have adopted this process definition format. However, a perspective on adopting a generic workflow model representation is too narrow in our view. Multiple process representations can be useful at different stages of process design, which need to be leveraged. For example, Petri nets are ideal for simulation and verification purposes because of their formal properties for a given process model. On the other hand, logic-based approaches are more suited to support process model generation based on their reasoning capabilities, in addition to providing flexibility and adaptability. Also, the idea of predefined workflow model (schema) or process definition indicates enforcement of rigid structure of processes that do not offer the opportunity to adjust the sequence or content of activities to correspond to the actual situation during execution. We illustrate these points in Section 2.6.4.

Also, the WAPI interface specifications define a set of low-level protocols for synchronously and asynchronously exchanging workflow data between various tools and the WES. While this approach is a widely accepted approach in a client-server environment, a consistent data format is difficult to obtain from different domains. Conformance becomes an issue if the workflow representations cannot easily convert their process data in alignment with this underlying model. A modular or layered approach is desired for interoperability between different workflow process represen-

tations [136].

In addition, the restriction of business processes is an unnecessary limitation on the application of workflow technology, since WFMS can also be applied to automation of software processes [157]. The terminologies adhered to by WfMC reference model are primarily related to activity-based process modeling (input-output-process type approach), allowing minimal support for alternative process representations such as those supporting constraint-based [390] or speech-act based approaches [261]. Similarly, the definition of workflow model, indicates the emphasis on support for individual activities. The support for group activities is restricted to worklist items which can be allocated to a potential pool (group) of participants, one of which choses to perform the work individually.

Other standardization efforts in the area of workflow modeling undertaken by organizations such as Business Process Management Initiative (BPMI), Organization for the Advancement of Structured Information Standards (OASIS), World Wide Web Consortium (W3C), and Object Management Group (OMG) overlap significantly with the WfMC proposals. Some of these efforts are summarized after the discussion on process modeling in Section 2.9.2.

2.6.4 Lifecycle of a Workflow Schema

To illustrate the contemporary workflow management approach, we consider the lifecycle of a workflow schema (process definition). Figure 2.8 illustrates the lifecycle of a workflow schema for a business process, through design and deployment (based on discussion in [395, 396]).

The lifecycle begins when business process analysts acquire and structure organizational processes into a workflow (Step 1). Defining a workflow schema involves identifying the different tasks that constitute a business process in an organization and then specifying the execution sequences along with the executing agents, control

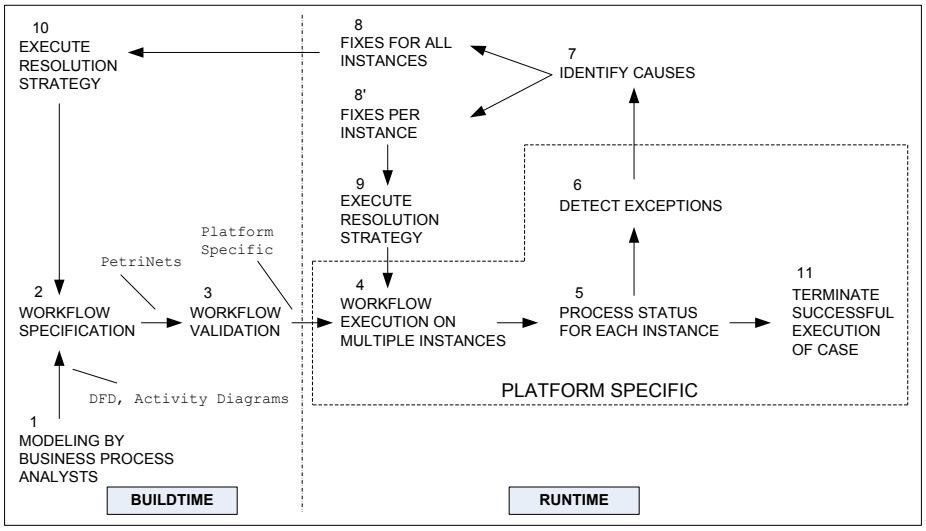


FIGURE 2.8. Lifecycle of a workflow schema for a business process

and data dependencies for the various tasks. The workflow thus defined is verified, tested and then executed (enacted) in a workflow engine (Steps 2, 3, and 4). A single workflow schema may cater to multiple workflow instances (or cases). Each executing case is monitored and tracked (Steps 5 and 6). Cases may execute normally without any errors or may lead to exceptions of various kinds. These exceptions may have a variety of causes and are handled by manual intervention (Step 7). The failures of the workflow cases are usually caused by underspecified or erroneously defined workflows during the modeling task, as it is usually difficult to acquire comprehensive knowledge to model all possible variations of a business process. Further, the workflow implementation environment changes as the business evolves; organizational roles and task assignments change. Repairs may be applied to a single executing instance or organizationally it may be decided to update the initial workflow specification (Step 8, 8′). Manual intervention requires knowledge of the business processes as well as the underlying workflow technology. The process of updating single instances and generic definitions is a highly knowledge intensive task in organizations because of the complexity of the workflows (Steps 9 and 10). The workflow validation/verification step is

usually done via simulation/animation and cannot identify all the errors early because of the large number of test cases that need to be checked.

The lifecycle in Figure 2.8 also indicates the different process representations that may be used during the various stages of the lifecycle. For example, Data Flow Diagrams (DFDs) and Unified Modeling Language (UML) Activity diagrams may be used by business process analysts, which may be then encoded into a specification. Such a specification may be translated into a Petri net based formalism for analysis. A review of existing process modeling formalisms is presented in later sections. Note that the activities in the figure are classified into build time (design) and run time (execution) activities. Runtime models are highly dependent on the technology platform used to execute a workflow. Thus, the process of deploying a workflow model into production requires a consistent set of model transformations that need to occur through different stages of the lifecycle. It can be noted that failure and change management approaches to support adaptive and dynamic processes are largely ad hoc. Once a workflow is deployed, maintenance of the workflow schema (or its instances) is manual and resource intensive [69]. Lack of proper process design guidelines can lead to different models (and instances) being updated in an inconsistent manner.

2.6.5 Workflow Management Systems to Business Process Management Systems

Instead of marketing their products as WFMS, recently workflow vendors are using the label Business Process Management Systems (BPMS). While there is certain amount of hype in using new labels, from functional and technical standpoints, certain points can be noted that distinguish BPMS from WFMS. Simply stated, we can consider BPMS as the next evolution of WFMS. BPMS propose to add robust application integration, application development, process analysis, and richer process simulation and modeling capabilities that traditional workflow systems are lacking. For example, consider the interfaces 2 and 3 from the WfMC reference model, rela-

beled as WAPIs for application integration, These interfaces are offered in C and do not follow the prevailing component models such as J2EE, COM/.NET, or CORBA. Similarly, WAPIs provide limited functionality when it comes to Enterprise Application Integration (EAI) tools. BPMSs promise to deliver sophisticated functionality for EAI, including mapping from one application format to another, through technologies such as messaging, component interfaces, and web services [65].

In the area of process design, WFMS does not offer tools that can help in process design and analysis, besides basic process modeling capabilities. Process modeling capabilities have recently been introduced. When they are introduced, they are included in BPMS or WFMS–relabeled–BPMS products. Similar trend applies for process analytics such as audit logs for process improvement. In addition to these enhancements, BPMS is supposed to be more flexible when reacting to organizational and process changes. The traditional WFMS have been criticized for their inflexibility. BPMS take the approach of providing a platform for developing and executing process solutions rather than WFMS, which exist more as an application to connect activities needed for a business process [65].

In essence, we can view BPMS as an evolutionary technology built with the foundations of WFMS and EAI, providing robust process coordination and management. In the context of this work, we use the terms WFMS and BPMS interchangeably, with these notions in mind.

2.7 Groupware Technology: GSS

Organizational processes routinely consists of tasks and problems that no one individual has the information and experience to solve them alone, so group tasks involving meetings have become a ubiquitous feature of modern organizational life [99]. Collaboration is thus essential for organizational value creation [295, 301]. Typical examples of group tasks include, but are not limited to, communication, planning, idea genera-

tion, idea organization, problem structuring and solving, issue discussion, knowledge elicitation, negotiation, conflict resolution, systems analysis and design, and collaborative group activities such as document preparation and sharing. While team efforts can be productive and successful, group work is fraught with challenges that can lead to unproductive processes and failed efforts [300, 98] (see Figure 2.10). Difficulties of teamwork range from poor preparations to vague follow-through, from loud talkers to free riders [296] (see Figure 2.9). Moreover, managers and knowledge workers spend a significant proportion of their time working in groups. Estimates of this proportion range from 60-70% for information systems (IS) managers to 30-80% for general managers [268]. This has motivated research in IS since the 1980s, which has led to the emergence of an entire class of information systems with focus on using IT to support group work, namely Group Support Systems (GSS). A brief overview of this research is given below.

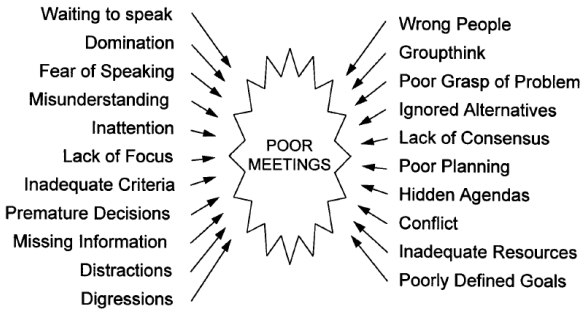


FIGURE 2.9. Meetings are difficult (based on [296])

2.7.1 Evolution of GSS Research

GSS research initiated out of the research stream of both Decision Support Systems (DSS) [17, 195, 358] and CSCW [91, 154]. Group Decision Support Systems (GDSS) emerged from DSS technology to support group decision making situations [215]. As stated by DeSanctis and Gallupe [93], GDSSs are "an interactive computer-based system which facilitates solutions to unstructured or semi-structured tasks by a set of

decision makers working together as a group." GSS can be considered as a superset of GDSS where there is no explicit assumption that the group will engage in decision making [91].

CSCW is the second stream of research and development, which contributed to the initiation of GSS field of work. CSCW is sometimes referred to in the literature as Computer Mediated Communication Systems (CMCS) or Computer Supported Communication (CSC). According to Hiltz and Turoff [179], these systems "use computers and telecommunications network to compose, store, deliver, regulate, and process communication among group members and between the computer and the group." CSCW systems differ from GDSS systems in two ways. One, they tend not to include any process support or decision support software; they are merely channels which support unstructured communication. And two, while they can be implemented within a same time and same place environment, they are much more commonly implemented across either time or place or both [91]. Research in CSCW is primarily seen as an offshoot of the office automation research efforts in the mid-1980s as discussed in Section 2.5.1 [364]. As overspecialization of work procedures limited the adoption of office automation systems, research shifted to less constrictive models that encouraged collaboration, encompassed under the label CSCW. A broad listing of technologies fall under the umbrella of CSCW. The most successful have been the email and blackboards. Modern day technologies in this arena include Lotus Notes, which build on the central theme of organization and presentation of electronic messaging [222]. Also, synchronous technologies such as video conferencing, shared whiteboards, and internet telephony have superseded the simpler technologies such as chat [153, 155].

GSS lies at the intersection of both these research streams, in that it combines the task-orientation of GDSS and the communication-orientation of CSCW. With advancements in research and technology, the boundaries between CSCW and GDSS have blurred over the past decade. GSS is also referred to as Electronic Meeting

Systems (EMS) or simply Groupware. Dennis et al. coined the term EMS to refer to support teamwork [91]. The EMS concept originated in work (PlexCenter) to support systems development teams for the PLEXSYS project [213, 212].

2.7.2 Group Support Systems

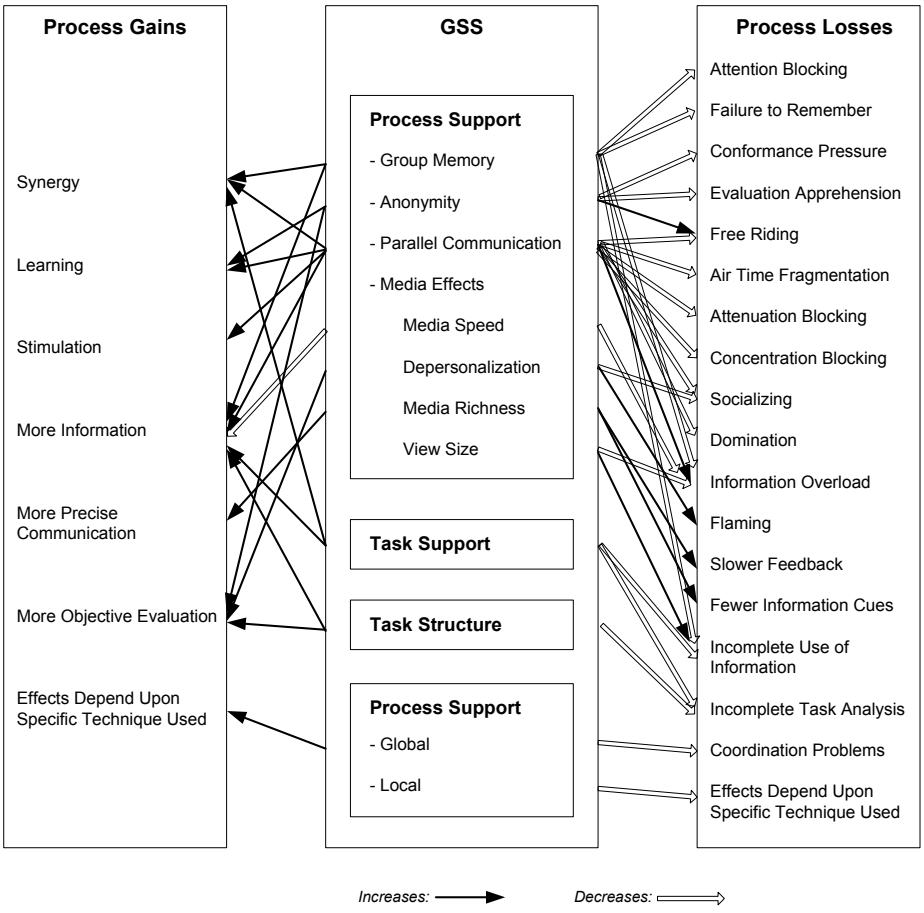


FIGURE 2.10. Group process gains and losses as GSS effects (based on [300])

GSS is defined as a computer-based information system to support intellectual collaborative work that consists of networked computers, special software, and typically a public screen [191, 300]. GSS provides techniques, software, and technology designed to focus and enhance communications, deliberations, problem solving, and

decision making processes of groups [294]. Group tasks that can be supported by GSS include, but are not limited to, communication, planning, idea generation, problem solving, issue discussion, negotiation, conflict resolution, systems analysis and design, and collaborative group activities such as document preparation and sharing [91]. Typical examples of GSSs include GroupSystems from University of Arizona [91, 373] and CoLab from Xerox PARC [359].

Collaboration in high-value organizational group tasks requires rich tool sets, structures, and support to assist the human interactions needed to accomplish goals and complete work [300, 55]. Team members may need to organize and synthesize ideas, generate and evaluate proposed alternatives, plan a course of action and carry out that plan in order to accomplish their shared goals [94, 300]. Due to the volume of information and the dynamic and complex nature of tasks, group may need a knowledge management process and an integrated environment to support that process [94, 300, 328]. GSSs aim at providing such collaboration process support by empowering groups through anonymity, equal participation, reduced domination, structured processes, and group memory capabilities, which may directly effect processes and outcomes [296]. These process support features are often unique to GSS and missing in other type of generic groupware tools [296].

GSSs such as GroupSystems can create an environment to directly impact and change the behavior or groups to improve their effectiveness, efficiency, and satisfaction, through supporting four fundamental mechanisms, namely, process support, process structure, task support, and task structure [300] (see Figure 2.9). Process support refers to the infrastructure, media, channels, and devices, that facilitate communication among group members (e.g., electronic blackboard). Three potential methods of process support integration within GSS include parallel communication [91], group memory [347], and anonymity [374]. Process structure refers to process techniques or rules that direct the pattern, timing or content of an interaction (group level or individual level) (e.g., shared agenda) [93]. Task support refers to the information and

communication infrastructure for task related activities (e.g., information from previous meetings for improving the understanding of current problems). Such support can encourage utilizing more information and enable issues to explored more fully than would otherwise be possible. Task support can also provide technological support for building organizational memory [349]. Task structure refers to techniques, rules, or models for analyzing task related information to gain new insights (e.g., problem modeling, multicriteria decision making, stakeholder analysis, value chain technique). Integration of task structure into GSS may allow tasks to be decomposed into small manageable chunks that allow groups to focus effort and reduce information overload [300]. Research in the area of GSS has focused on various ways of supporting these mechanisms. In this dissertation, our focus on designing group tasks involves addressing issues related to process and task structure. We discuss these further in the context of our work in Chapter 6.

2.8 Evolution of Tools and Techniques for Systems Development

Process coordination mechanisms have evolved since the 1980's as discussed in Section 2.5. It is informative to also look at the evolution of system development techniques and situate this work, as shown in Table 2.5 [73, 72].

The first generation (1950–1960) of system analysis techniques (mainly portraying information flow) used to design computer-based systems evolved from process flow charts used by industrial engineers in the early 1900s. Analysts did not develop tools specifically for computer-related system analysis, but modified techniques used for analysis and design of manual systems.

The second generation (1960–1970) of system analysis techniques included decision tables, Accurately Defined System (ADS), Study Organization Plan (SOP), Honeywell's Business Information Systems Analysis and Design (BISAD).

The major distinction of the third generation (1970–1980) of system development

TABLE 2.5. Evolution of tools and techniques for systems development

Period	Phases	Tools & Techniques	Remark
1950–1960	First generation	• Information process chart • System flowchart	Modified manual systems techniques for computer-based systems
1960–1970	Second generation	• Decision tables • ADS, SOP, BISAD	Improved decision & documentation techniques for systems analysis
1970–1980	Third generation	• DETAB-X, automated ADS • Information algebra, PSL/PSA	Computer-aided systems analysis and design
1980–1985	Fourth generation	• PLEXSYS • BSP, BIAIT/BICS	System development systems Emergence of structured analysis
1985–1990	Fifth generation	• ISDOS	System design optimization Computerized system planning

techniques was the emphasis on computer-aided techniques, driven by the increased scope and complexity of systems to be computerized. Two parallel research efforts occurred. One concentrated on automating manual system analysis and design techniques (e.g., decision table processor DETAB-X by CODASYL, automated ADS). The other concentrated on building a theoretical basis for automation of systems development methodology (e.g., Information algebra). The results of these two activities were utilized later by another research effort, which focused on merging the earlier developments into a optimal set of computer-aided techniques (Problem Statement Language (PSL)/Problem Statement Analyzer (PSA), System Optimization and Design Algorithm (SODA)). The SODA/PSL was designed to express desired system outputs independent of processing procedures, what data elements these outputs comprised, and formulas to compute their values. The SODA/PSA accepted inputs in SODA/PSL, analyzed them for correct syntax, and generated an error-free problem

statement in machine-readable format, along with a coded statement for use in the physical system design process.

Computer-aided techniques continued to improve resulting in the fourth generation (1980–1985), which included systems development systems such as PLEXSYS (extension of PSL/PSA) and computerized systems planning techniques such as Business System Planning (BSP), Business Information Analysis and Integration Technique (BIAIT), and Business Information Characterization Study (BICS).

Integrated system development including logical system design, physical system design, system building, testing, maintenance and modification, became the focus in the fifth generation system development techniques (1985–1990), which included Information System Design and Optimization System (ISDOS) project at the University of Michigan.

It is interesting to reflect on these system development techniques by observing the trend towards automated system development using Computer Aided Software Engineering (CASE) tools. A parallel trend is evident in the process management domain. Process coordination is the goal for process management systems as opposed to systems development goal of CASE tools. This dissertation focuses on proposing computational techniques, which can automate process model generation, as well as support flexible and adaptive process coordination. In case of automated generation of process models, the objective is to provide computer-aided tools for the process designer to coordinate and manage a process, encompassing manual as well as automated activities. On the contrary, in case of CASE tools, the objective is to provide computer-aided tools for the systems analyst and designer to model the information system in terms of information process flows, leading to generation of code for the system development.

2.9 Process Modeling

Business Process modeling is intimately connected to organizational process design. Traditionally, process designers explore the design space (although implicitly) for seeking the right kinds of process models. Process modeling brings numerous advantages to an organization deploying the same, some of which are listed below [103]:

- the use of explicit process models provides a means for communication between managers and business analysts with respect to the structure of the business process, and the IT architects, software developers, and systems administrators with respect to design, implementation, and operation of the technical infrastructure supporting these processes,
- the emphasis on process models rather than code allows for better change management,
- the explicit representation of the processes supported by an organization allows their automated enactment, which in turn leads to increased efficiencies by enabling the reduction of redundant data entry tasks and providing opportunities for interconnecting otherwise separate transactions,
- the explicit representation of the processes enables management support at the (re)design level with a variety of verification, process performance evaluation and prediction tools, and
- the explicit representation of the processes enables management support at the control level with a variety of process execution analysis and control, and process mining tools.

2.9.1 Process Modeling Formalisms and Languages

A variety of process modeling languages are available for the specification of workflow models. Most process modeling methods are influenced by more than one discipline and often overlap with each other in some aspects. We now present some of significant developments in this area.

Business process modeling methods were initially inspired by process modeling techniques which provide precise formats to capture processes that are practiced in manufacturing environments. These techniques have been adapted and extended by business process modeling methods to capture and formalize processes practiced in non-manufacturing environments. We have earlier discussed (see Sections 2.4.2 and 2.6.3) the process modeling notions referred to in the Handbook of Organizational Processes [250] and the Workflow Reference Model [182]. Some of the related business process modeling formalisms and languages include Process Interchange Format (PIF) [226], Process Specification Language (PSL) [338, 159], Integration DEFinition Language (IDEF0) [281], UML's Activity Diagram (extension) [329, 304], Event-driven Process Chains (EPC) [324, 337], and Petri nets [276, 144, 105]. Also, simulation techniques often used in business process modeling are based on techniques used in the manufacturing environments [313, 312].

With increasing need to promote understanding and interoperability of process semantics, new process languages and standards are being proposed by various organizations. Research in area of semantic web[1] [22] has been instrumental in characterizing these languages with representations based on XML, RDF [311], or OWL [220].[2] They may also provide constructs to assist communication between processes over the Internet. Examples of such languages are Web Services Business Process Execution

[1]Semantic Web is a (conceptual) layer on the Internet consisting of data and applications where meaning (or semantics) is encoded to support knowledge sharing.

[2]XML, RDF, and OWL are representational languages that have been designed to describe the semantics of data and to support machine processing (vs. human processing)

Language (BPEL) [19], ebXML Business Process Specification Schema [232], XML Process Definition Language (XPDL) [400], Business Process Modeling Language (BPML) [24], OWL-based Web Service Ontology (OWL-S) [258].

Software engineering is another area that has been influential in the development of process modeling techniques. Providing software engineers with clear system requirements and directions for building a better IT system were the main motivating factors for using process modeling by the software engineering community. Process modeling for software systems development provide the means to describe a business process from a higher level of abstraction, capturing its operations from a management standpoint, not confined by technical, specifically IT considerations. Implementation independent approaches such as this provided methods and techniques such as Role Activity Diagram (RAD) [305], Meta-model by Scacchi et al. [263], Swim-lane Diagram by Rummer et al. [330], ORDIT [96], BSDM's business modeling method [186], the business modeling approach using (extended) UML notation by Eriksson et al. [112] and by Rational [185], and the reengineering method developed by Jacobson [188].

Some process modeling methods focus on capturing and tackling the wide organizational issues within a business. These include methods which capture the functional, structural, and/or cultural aspects of an organization. In this context, from a knowledge management perspective, Macintosh et al. [240] and Schreiber et al. [339] provide a framework to identify, obtain and maintain the required knowledge and skills for an organization and means to make use of them to achieve organizational objectives. Yu et al. provide graphical notations to capture business strategies and their rationale in the Strategic Dependency and Strategic Rationale Models [416]. To promote better communication via a common language within and between organizations, ontologies have also been developed for businesses. Representative examples of work in this area are the Enterprise Ontology developed by Uschold, King, Moralee and Zorgios [372] and Tove by Fox, Grüninger et al. [158].

Carlsen [61] propose the following classification of process modeling approaches :

- *Input-Process-Output (IPO)-based Languages*, such as the activity networks used in IBM Websphere MQ (formerly known as MQ Series) [233]. These languages describe a workflow as a directed graph of activities, denoting the sequence of their execution.

- *Speech-Act-based approaches* (sometimes called Language Action approaches) as used in Action Technologies' ActionWorks Business Process Manager [411]. These approaches model a workflow as an interaction between (at least) two participants that follow a structured cycle of conversation. Namely the phases negotiation, acceptance, performance, and review are distinguished [261].

- *Constraint-based modeling methods* such as Generalized Process Structure Grammar (GPSG) [145]. These approaches describe a process as a set of constraints, leaving room for flexibility that is otherwise governed by the restrictions of the IPO- or Speech-Act-based languages.

Process definition languages (PDL) for the specification of workflow models can be separated into languages with a graphical representation and languages with a textual representation. Graphical modeling languages are typically found in process modeling tools, where they are used for specification of the overall process structure and the decomposition of sub-processes and activities. Most of these languages are focused on modeling business processes from functional and behavioral perspectives. An overview of graphical and text-based process definition languages, based on zur Muehlen's work [422], is given in Table 2.6.

PDL approach to process modeling can be viewed as programmatic or procedural approach, whereas constraint-based GPSG approach to process modeling can be viewed as a declarative approach. While a PDL-driven WFMS relies on predefined rigid process structures, GPSG-driven WFMS can be more flexible by allowing the

TABLE 2.6. Process definition languages (adapted from [422])

	Textual Representation	**Graphical Representation**
Graph-based Languages	• Workflow Process Definition Language (WPDL, XPDL) [399, 400] • Business Process Modeling Language (BPML) [53] • Business Process Execution Language (BPEL4WS) • EPC Markup Language (EPML) [262]	• Activity Nets [233] • Business Process Modeling Notation (BPMN) [402] • Control Flow Graph [13, 15] • Event-driven Process Chains [196]
Net-based Languages	• Petri Net Markup Language (PNML) [49] • Yet Another Workflow Language (YAWL) [385]	• Funsoft Nets [156, 90] • Flow Nets [106] • Workflow Nets [379]
Workflow Programming Languages	• Mobile [187] • FlowMark Definition Language (FDL) [233] • Transaction Datalog [51]	• State and Activity Charts [167]

TABLE 2.7. PDL-driven versus GPSG-driven approaches(adapted from [145, 422])

	PDL Approach	**GPSG Approach**
Workflow Engine	PDL grammar • Parser of user-defined process models • Interpreter of process models • Lexicon: activities, dependencies among activities	GPSG grammar • Generator of user-defined process models • Constraint solver • Compiler of processes
Workflow Model	• Legal phrase defined by the user respecting the PDL grammar	• User-defined process grammar • Lexicon: activities, documents, dependencies defined as feature constraints
Workflow Instance	• Instantiation of workflow model	• Legal phrase generated by the user from the process grammar
Flexibility in Workflow Instance	• Conditional statements in workflow model • Change of the workflow model	• New legal phrase in the process grammar • New phrase in the modified process grammar

process designer to to specify his own language (lexicon), including the objects that are part of the process [422]. As pointed out by Glance et al. [145], "using a generative grammatical approach, a given process instance can be incrementally singled out from the space of possible workflows defined by the rules as the process evolves." The difference between PDL-driven and GPSG-driven approaches is outlined in Table 2.7. Declarative process modeling approaches in the literature such as GPSG have motivated the approach we present take in this work.

TABLE 2.8. Standards for process modeling languages (part 1)(adapted from [422])

Name	**PSL**	**PIF**	**GSPG**
Origin	NIST Standardization	MIT Research	Xerox Research
Specification	KIF [137]	KIF [137]	Text
Notation	No	No	No
Objective	Modeling	Model Exchange	Modeling
Tool Support	Prototype	Prototype	Prototype
Area	Workflows	Workflows	Workflows
Source	Knutilla et al. [205] Note: Merged with PIF	Lee et al. [226] Note: Merged with PSL	Glance et al. [145], Dourish et al. [97]

TABLE 2.9. Standards for process modeling languages (part 2)(adapted from [422])

Name	**UML**	**WPDL/XPDL**	**BPML**
Origin	OMG Standardization	WfMC Standardization	BPMI.org Standardization
Specification	MOF, UML, Text	EBNF, Text, XML	XML Schema
Notation	Activity Diagrams	No	Yes, BPMN [402]
Objective	Modeling	Model Exchange	Modeling
Tool Support	Product	Product	Prototype
Area	Workflows	Workflows	Workflows
Source	OMG [151], DSTC, PrismTech [100]	WfMC [399, 400]	BPMI.org [53]

2.9.2 Standards for Business Process Modeling Languages

With the growing number of modeling languages and formalisms being proposed, there is increasing confusion in choosing the right ones for the purpose at hand. Moreover, several standards for these languages are also being put forth along with mergers within different standards. Consequently, the role of process designer is becoming increasingly challenging. An overview of current standardization efforts for process modeling languages in the context of workflow mananagement is given in Tables 2.8, 2.9, and 2.10.

TABLE 2.10. Standards for process modeling languages (part 3) (adapted from [422])

Name	**BPEL4WS**	**WSCI**	**WSCL**	**BPSS**
Origin	IBM, BEA, Microsoft, SAP, Siebel (OASIS)	Sun, BEA, Intalio, W3C Note	HP Labs Research	ebXML (OASIS) Standardization
Specification	XML Schema	XML Schema	XML Schema	XML Schema
Notation	BPMN (planned) [402]	No	No	No
Objective	Modeling	Modeling	Modeling	Modeling
Tool Support	Product	None	None	Product
Area	Web Services	Web Services	Web Services	Web Services
Source	IBM [19]	W3C Note [37]	W3C Note [25]	ebXML [104]

2.9.3 Workflow Patterns for Comparing Process Modeling Formalisms

Recently, an initiative started to systematically compare features of WFMSs and evaluate the suitability and expressive power of workflow modeling languages. The core of this initiative includes a set of control flow patterns, cataloged by van der Aalst et al. [384], originating from commonly recurring business requirements. The patterns are grouped in six main categories, as shown in Figure 2.11.

The basic patterns capture elementary control flow: a sequence of activities in

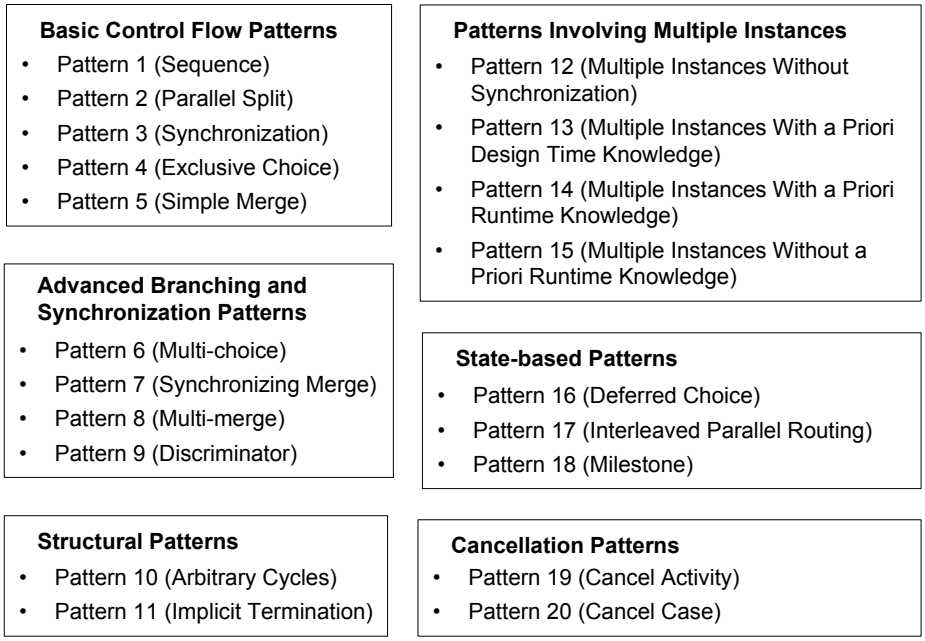


FIGURE 2.11. Overview of 20 workflow patterns described in [384]

which two activities execute after each other (meaning that the second one becomes enabled for enactment when the first one completes); a parallel split (AND-split) and parallel synchronization (AND-join) of activities in which several activities execute in parallel (or in any order) and are later synchronized (the synchronize activity waits until all parallel branches are completed, and then continues); and an exclusive choice (XOR-split) and exclusive synchronization (XOR-join) in which one out of several branches is chosen based on some condition and later synchronized (the synchronize activity waits until the active branch completes, since only one branch is assumed to be active).

The advanced branching patterns capture more complicated branching scenarios and are often not directly supported in existing systems: the multi-choice (OR-split) in which one or more out of several branches is chosen based on some condition; such a split can be joined in three ways: the synchronizing merge waits for all active branches of the split, and proceeds only after they have completed; the discriminator is similar,

but the first active branch that completes already triggers the activity following the discriminator, all other active branches that complete later are ignored; the multi-merge joins the active branches without synchronization, the activity following the multi-merge is started once for each active branch.

The structural patterns capture modeling situations that are usually forbidden in workflow management systems: arbitrary cycles are unstructured loops without predefined entry- and exit-points; implicit termination captures that a process should terminate implicitly when there are no activities left to perform.

The patterns involving multiple instances capture situations on which one case in a workflow has several child cases that are being instantiated in parallel. Each of these child cases needs to complete before the parent case can continue. The problems relate to being able to instantiate child cases from a parent case, and to being able to synchronize these instances and continue with the parent case after all child cases complete. For synchronization the number of child cases is relevant, this number can be determined at design time or at runtime. If determined at run time it can be fixed before the instantiations start, or it can dynamically change during execution of the child cases.

State-based patterns involve scenarios where an explicit notion of the state of the workflow process is relevant: the deferred choice is a split in which one out of several branches is chosen not by the workflow management system but by the environment; all branches are offered to the environment and the first one to be chosen invalidates the others; the interleaved parallel routing is an advanced sequential pattern, in which the order of the activities is arbitrary but the activities are not executed in parallel; the milestone pattern captures that an activity becomes enabled after completion of some activity but that it becomes disabled again if some other activity starts.

Cancellation patterns involve retracting running cases or executing activities: the cancel activity captures that some running activity can be canceled by another activity or some external event; the cancel case captures that the whole running case can be

canceled by the execution of some activity or some external event.

These patterns address the behavioral perspective of process modeling languages, as identified by Jablonski et al [187]. It has to be noted that these patterns do not describe any minimal business requirements, they are a systematic summary of available features in process modeling languages and existing systems. An open research question in this regard is which of these patterns are essential and necessary for a usable workflow management system. We use these patterns as an evaluation platform for our proposed process modeling technique.

2.9.4 A Critical Look at Some Existing Process Modeling Formalisms

Petri Nets : Historically, Petri nets have been around since the sixties [306] and have been extended with color [189, 190] and time [256] to improve expressiveness. Petri nets where tokens carry data (i.e., are colored) are often referred to as high-level Petri nets and are supported by tools such as CPN tools [316], and ExSpect [370].

Many researchers have advocated the use of Petri nets for workflow modeling, e.g., [420, 110, 187, 141, 380, 334]. Wil van der Aalst mentions three good reasons for using Petri nets [378]: their formal semantics, the fact that they are state-based instead of event-based, and the abundance of analysis techniques available.

These advantages make Petri nets a good candidate for a simulation tool in the realm of business process design. However, from a design perspective, Petri nets have several limitations. As pointed out by Eshuis and Dehnert, the standard token-game semantics of Petri nets is not completely suitable for modeling of workflows [117]. Although Petri nets are said to be state-based, these states are also based on token semantics. Modeling and reasoning about the dependencies of tasks, using task attributes and data entities becomes cumbersome with increasing complexity of the workflow models. The token-game semantics of Petri nets models closed active systems, whereas WFMS are actually open, reactive systems [118, 116, 117]. The re-

sulting underlying problem is that the distinction between the environment and the WFMS is not captured. Events can modeled as places or transitions. In the former case, a place is created for each input event, in which the environment is supposed to place a token when an event occurs. The problem is that event broadcasting is not possible since transitions consume these event tokens, which also makes cancellation events problematic to model. In the later case, a transition is created for each event. Synchronization is needed between transitions that model tasks and those that model events, to infer if a transition is triggered by the occurrence of an event.

With respect to the control flow patterns, Aalst has noted serious limitations of Petri nets when modeling (1) patterns involving multiple instances, (2) advanced synchronization patterns, and (3) cancellation patterns [383]. In the first case, in an example such as a customer ordering multiple items, it is necessary to model a variable number of instances executed in parallel. While high-level Petri nets can theoretically support this, the designer has to keep track of the number of active instances still running along with instance identities and synchronization issues. In the second case, it is hard to model processes, where some optional flows need to be synchronized, using Petri nets. For example, in a process of booking a business trip involving booking a flight, hotels, rental car, etc., some activities can occur in parallel. This means that one trip may involve only a flight, another trip may involve a hotel and a rental car, and so forth. The problem faced with Petri nets is that is not clear which subflows need to be synchronized. Finally, in the third case mentioned, quite some bookkeeping is necessary with Petri nets to remove tokens from an arbitrary set of places in order to withdraw an instance, or a subprocess or the entire process. This can be attributed to the local nature of Petri net transitions, which makes such reactive changes difficult to handle.

From a flexibility point of view, Petri nets are quite rigid, since the entire process model is planned ahead of time – complete with alternatives – by the designer with ensuing analysis before deployment to WFMS [142, 145]. However, any changes to the

processes (such as addition, removal of tasks) must reflect in changing the process model, thus rendering the previous analysis not useful. This is a costly and time consuming restraint on using Petri nets in dynamic environments.

According to Wolstenholme et al. [415], "the technique's limites range of primitive analytical constructs compels the analyst to adopt a specific (usually high-level) approach which can sometimes limit the scope of analysis achievable.

In summary, it is noted that Petri nets are good for simulation and analysis of tasks (including performance analysis) and planning of resource assignment to tasks and for business processes in particular [202]. While classical Petri nets are limiting, high-level Petri nets can allow modeling of information and data, although reasoning capabilities are a big limitation.

π-Calculus : π-calculus [266, 265] is a process algebra variant, recently being proposed by the Smith and Fingar [354] as a formal foundation for workflow. It extends the Calculus of Communicating Systems (CCS) with notions of mobility, which includes communication and change. Communication takes place between different π-calculus processes. The structure of the processes changes over time by communication, e.g., a process can dynamically include other processes which it received through communication.

The main difference between Petri nets and process algebra is that Petri nets are based on (bipartite) graphs while process algebras are based on a textual (i.e., rather linear) description. Many notions developed for Petri nets have been translated to process algebra and vice versa, although fundamental differences remain. For example,the notion of invariants developed for Petri nets does not exist in process algebra [318, 319]. See [42] for a detailed comparison of Petri nets and process algebra.

An ongoing debate amongst researchers is focused on questioning the additional advantages of π-calculus over Petri nets, especially in the context of Web Service Composition Languages (WSCL) [382]. One camp has been advocating that workflow

languages BPEL4WS, BPML, WSFL, XLANG, XPDL,and WSCI have been based on π-calculus [355]. However the other camp has refuted this claim and challenged the other camp to prove their point with real world examples [381]. While there is no conclusion to this debate, it is certainly reflective of the confusion and lack of formal semantics in the business process modeling domain.

Based on the execution semantics of π-calculus, the behavior of each of the workflow pattern has been documented in [314]. While it has not been shown how mobility can actually enrich the workflow domain, requirements like flexibility and reaction are noted to be challenging [354]. Since the π-calculus was designed to model such highly dynamic systems, it is hoped that it might offer new ways to face the challenges in the workflow domain.

UML Activity Diagrams : UML activity diagrams are special cases of UML state diagrams, which in turn are graphical representations of state machines. The state machine formalism as defined in UML, is a variant of Harel's statecharts [169]. The suitability of statechart-based notations for process design has been noted by some studies [277].

While activity diagrams are comparable to other existing process modeling formalisms, their commonly noted drawback is that their syntax and semantics are not fully defined. It is observed that the features inherited from Harel's statecharts [169] have a formal operational semantics, while the features specific to activity diagrams are only partially formalized in the standard.

In essence, as is noted by Dumas et al. [102] and Wohed et al. [414], the formalization of the activity diagrams notation, and the evaluation of its suitability for workflow design are still open issues. This is due to the inherent difficulties in assessing a language that does not have commonly agreed upon formal semantics not an execution environment.

Summary : We conclude this section with the typical issues faced by the process designer in developing robust workflow models using existing process modeling formalisms. (1) The business process may be highly complex and the process knowledge provided by analysts may be highly fragmented, unstructured, incomplete, and possible inconsistent. Further, since multiple analysts gather such knowledge independently, processes may be locally consistent, but globally inconsistent. Current formalisms implicitly assume requirement completeness, which leads to inflexibility in change management situations. (2) Due to the rigidness of the formalisms, the modeler needs to have knowledge of the specific requirements for, and variations of a business process at different stages of its lifecycle to account for all exigencies that may occur during the process lifecycle. (3) Developing a process flow (by resolving concurrent and sequential intertask dependencies) is currently confounded by the issue of scheduling and resource allocation. Task dependencies due to domain causality are not distinguished from resource related dependencies. Such confounding results in problems during workflow re-design (after failure). Additionally, buildtime scheduling decisions may not be applicable during runtime. (4) Validation is primarily simulation-based. It is like to be incomplete (considering the cost of scenarios to be analyzed) and many types of errors may go undetected. (5) Runtime failure management is complex because of the inherent inconsistency between the build time model and its implementational counterpart. Execution constraints may require business process changes or vice versa. The current coupling between the design and enactment representation is very loose and implementation dependent. No explicit state information either of the environment or the current execution is maintained to correlate the actual behavior of the system with that envisioned during the design phase. Many current WFMS capture audit information, however, use of such information for adaptivity or runtime reactive control is minimal [422]. (6) Change management of a workflow model may be triggered at any stage of the lifecycle. Existing formalisms offer very little in terms of support for propagating changes, minimizing side-effects

and developing effective recovery strategies (from an application perspective). (7) Current managerial approaches to workflow modeling perceive it as a programming activity rather than as an organizational process design and knowledge management activity at a higher level of abstraction [120]. Thus, resources allocated to workflow design and expectations of tool capabilities to support the same are limited. (8) Existing process modeling formalisms do not differentiate between individual and collaborative activities or have ways to represent group interactions in the process design phase. This limits their capabilities to modeling individual activities, which propagates to workflow systems implementing these process models.

2.10 Conclusion

Organizational theory and coordination science plays a major role in process design. They outline the need for flexible and adaptable process models to make process coordination tools more effective. They also indicate the need of an integrated view of individual and collaborative activities. The study of existing process support tools such as WFMS, groupware as well as existing modeling formalisms, illustrate the need to address the research questions mentioned in Chapter 1 and the lay the ground work for our proposed approach to process design, described in the following chapters.

Chapter 3
Research Methodology

We base this research on two related methodologies, which are often used as a benchmark to conduct research in the MIS domain, namely, systems development methodology, and design science paradigm. These are briefly explained in the sections below, followed with discussion on application of these principles in the context of this dissertation.

3.1 Systems Development Methodology

The University of Arizona has been instrumental in proposing, refining, and promoting the systems development research methodology for the past several years. It was put forth by Nunamaker et al. [299, 293] and is a multimethodological approach consisting of four research strategies: *theory building*, *experimentation*, *observation*, and *systems development*.

Theory building includes development of new ideas and concepts, as well as conceptual frameworks, new methods or models. The relevance of theory building is that it contributes to the body of knowledge in the application domain, providing potential insights and impacts on practical applications, as well as guiding the other three research strategies.

The main goal of experimentation is to bridge the gap between theory building and observations. It can include validation of the underlying theories or models, or deal with issues of technology acceptance. This methodology can be used to refine models or theories and also to improve systems.

Observation includes other research strategies such as case studies, field studies.

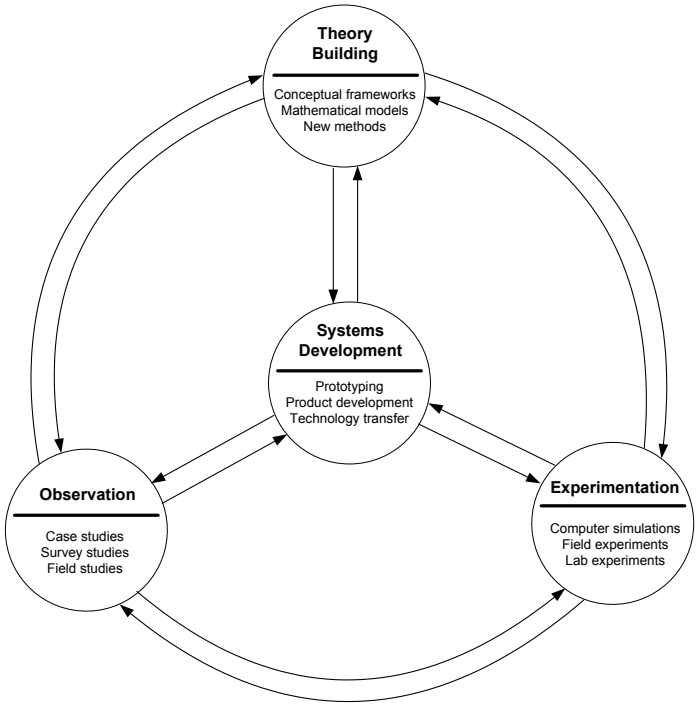


FIGURE 3.1. Systems development methodology [299, 293]

The goal is to get a general feeling for what is involved in the research domain.

Systems development involves concept development, system architecture, prototyping, product development, and technology transfer [76]. Initial research with newer ideas typical stops at prototyping and may continue later, incumbent upon favorable conditions in the industry. It has been emphasized that systems development be used along with other research methodologies to gain deeper understanding of a complex research domain.

One of the unique features of this methodology, it that it was the first IS methodology that supported systems development to be a perfectly acceptable piece of evidence (artifact) in support of a proof, where proof is any convincing argument supporting a worthwhile research question or hypothesis. In this sense, systems development is viewed as "proof-by-demonstration". In this context, Nunamaker et al. [299] state "(...) the developed system serves both as a proof-of-concept for the fundamental

research and provides an artifact that becomes the focus of expanded and continuing research."

As shown in Figure 3.1, systems development forms the hub of research that interacts with other research methodologies to form an integrated and dynamic research paradigm.

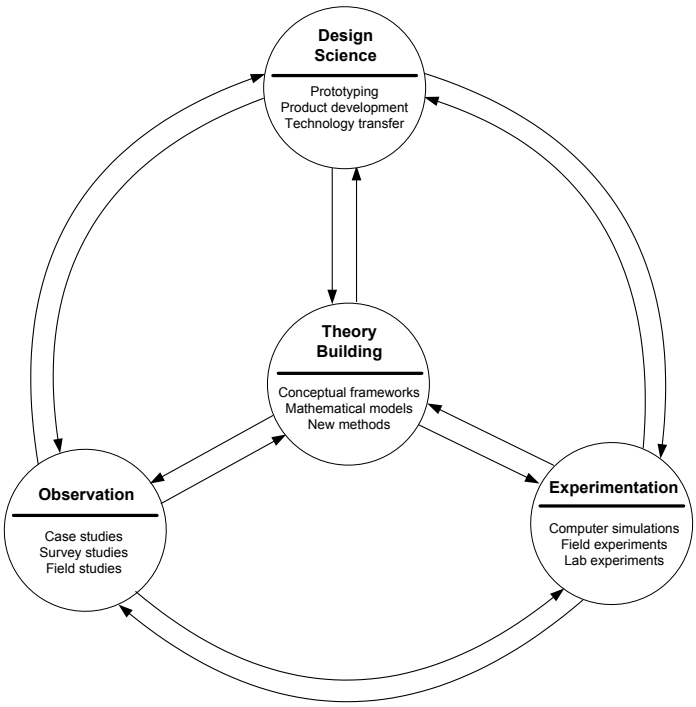


FIGURE 3.2. Modified systems development methodology [292, 299]

Recently, Nunamaker has proposed an alternative view of this methodology with design science replacing systems development and theory and model building forming the hub of research [292] (see Figure 3.2). This was in response to design science research methodology, lately proposed by Hevner et al. [177]. The goal was to initiate a panel discussion on the following three questions [292]: (1) Is "design science" another name for systems building in computer science and engineering? (2) Is system building the "central focus" of information systems research? (3) Is design science a valid research methodology? In was discussed that in many ways systems develop-

ment methodology and design science approach are close and propose similar research approaches under different labels.

3.1.1 Application of Systems Development Methodology

For this research, we employ the systems development methodology as follows.

The theory building phase involves the proposed Structured Process Design Cycle (SPDC) as a overall framework for organizational process design (Chapter 4). We have also used model development at other stages of this research. We've explored the application of situation calculus and AI planning formalisms, for proposing novel ways to model flexible and adaptable business processes (Chapter 5). We've also modeled collaboration process patterns, and detailed them to allow designing of group activities (Chapter 6).

The systems development phases involves the design support tools, for generation of process models embedding AI planning methods (Chapter 5) as well as for designing collaborative tasks (Chapter 6). Each of these tools have gone through the concept development, architecture, and prototyping phases. We demonstrate their utility with examples in this dissertation as a proof-of-concept.

The experimentation and observation phases involve validation methods employed and interpretation of their results. We have used case studies for demonstrating the feasibility of models and the system in the context of specific application domains. Also, comparison based on standards (Chapter 5) as well as existing design processes (Chapter 6) allow us to make convincing argument in support of our research questions.

We discuss design science research in Section 3.2

3.2 Design Science Research

Design science paradigm [177] has its roots in engineering and what Herbert Simon, the 1978 Nobel Laureate in Economics, termed "the sciences of the artificial" [348]. It seeks to create innovations, or *artifacts*, that encompass the ideas, practices, technical capabilities, and products required to efficiently accomplish the analysis, design, implementation, and use of information systems [176, 253]. It is similar in theme to the design theory proposed by Markus et al. [255]. It is situated in the overall IS research framework (see Figure 3.3), proposing a synergistic approach in which design science creates innovative artifacts for specific information problems and behavioral science meaningfully engages these artifacts in the development of theory for anticipating their impact [177].

Thus, the artifacts are created with hope to extend the boundaries of human and organizational capabilities by providing intellectual and computational tools that address problems and tasks not previously thought to be tractable with IT support. They include (1) constructs, (2) models, (3) methods, and (4) instantiations [176]. Constructs provide the language and formalism to express the problem at hand. Models use these constructs to represent the design problem and its solution space in the application context, with the goal of better exploration of design decisions and changes in the real world. Methods define ways by which solutions to the design problem can be reached. They range from formal, mathematical algorithms to informal, textual descriptions of "best practice" approaches. Instantiations include specific examples of how to implement constructs, models, or methods in a working system. They demonstrate feasibility, enabling concrete assessment of the artifact's suitability for the intended purpose, as well as gaining insights about the real world applications [177].

As seen from the discussion on design science as well systems development methodology, we concur with Nunamaker's view [292] that design science paradigm is another

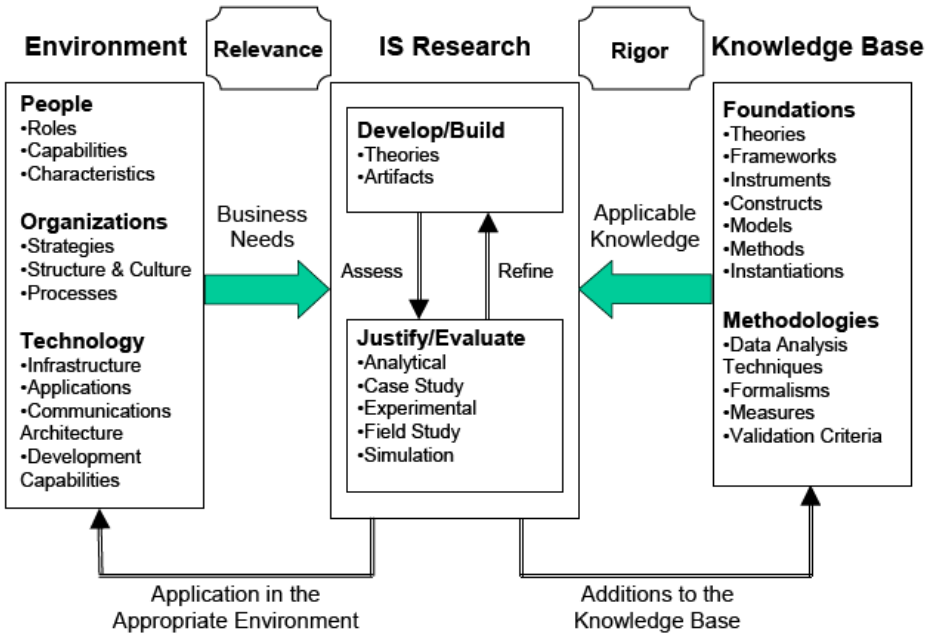


FIGURE 3.3. Design science as a part of IS framework [176, 177]

view of systems development research. Although design science has lately been proposed as a new research paradigm [177], situated in the overall IS research framework along with other methodologies, systems development has been carrying this viewpoint since much before, as can be seen with Nunamaker et al.'s system development methodology [299].

3.2.1 Application of Design Science Paradigm

This research has strong ties with design science from two perspectives. Firstly, as a reference research paradigm, it has helped guide the creation and evaluation of different artifacts, as noted in Section 3.1.1. Secondly, it has also guided our specific proposed approach to process modeling. While the details of this approach are provided in Chapter 5, we would like to point out our proposed process design approach is analogous to the design science approach, in that, it provides a formal description of activities, states, axioms, in the application domain (constructs), the process

models generated using these building blocks by searching of the process design space consisting of multiple possibilities (models), the use of AI planning algorithms as well as interleaving techniques to create flexible process models (methods), and finally the instantiation of this approach in different application domains (instantiation).

3.3 Conclusion

We have discussed two research methodologies, widely accepted in the MIS field, to base our research work on. The emphasis of both methodologies is on system and model development, along with instantiation and validation techniques. Following chapters describe our proposed approach, using these methodologies as guiding principles.

Chapter 4
Conceptual Framework

This chapter is focused on presenting the overall conceptual framework. We first explain the requirements that such a framework should support in Section 4.1. We then explain the rationale for our proposed approach in Section 4.2.1 following which the proposed framework outlining a new approach to organizational process design is presented in Section 4.2.2. Finally, Section 4.3 presents two application domains in which we validate the proposed approach.

4.1 Requirements of an Integrated Process Design Model

Based on the existing research in workflow process modeling (see Chapter 2), and noting the need for flexible workflow design, following requirements for an integrated process design model have been identified. These requirements guide our research approach for addressing the research questions mentioned in Chapter 1.

- *Process abstraction*: Processes should be modeled at different levels of abstractions. Hierarchical modeling of processes can allow for better process design with respect to accomplishing goals of different organizational processes.

- *Explicit process state representation*: In order to allow for mapping between design specification and execution of processes, state representation would be crucial. This can also allow to clearly track information flow between activities. Additionally, a priori behavioral reasoning during workflow design and dynamic control and redesign during enactment, with well-defined semantics, can be supported with explicit state information.

- *Generative process design*: A declarative and a generative approach to process design, which is computationally supported, can alleviate costly, time and resource-intensive manual process design.

- *Flexible process design*: An added advantage of generative process design can be to offer reactive support for dynamic and/or unexpected changes to processes, which is one of the core challenges faced by today's organizations.

- *Exploration of multiple process alternatives*: Searching the design space for alternative process models with clearly defined ordering constraints, can allow the process designer to look at various possibilities in which a particular business process problem can be solved. Evaluation of these alternatives based on optimization metrics can allow for better and optimal process model development.

- *Staged approach to sequencing, resource allocation, and scheduling of tasks*: Various attempts in the literature to consider these aspects at the same time have found to be difficult to manage with the complexity involved. Instead, a staged approach to considering these constraints can help alleviate this problem.

- *Tighter coupling between design, verification and execution*: Tighter coupling (from a semantic perspective) is necessary between specification, verification and enactment to allow for development of robust workflow schemas that can incorporate runtime constraints during enactment and facilitate feedback of execution knowledge to the design phase.

- *Integrated view of individual and collaborative activities*: A holistic view of supporting interleaved individual and team tasks in processes can avoid information losses that currently occur due to process fragmentation. Information generated in collaborative decision making activities can be relayed back to the overall process that this team activity was part of.

- *Structured design of collaborative activities*: Current groupware approaches, more or less solely based on communication support, need to be enhanced with process structure and support in order for the above mentioned integrated perspective to be effective.

4.2 Proposed Approach to Organizational Process Design

4.2.1 Rationale

The above mentioned requirements need to be captured in an overall framework in order to support process design and coordination activities, for a process designer. We begin by taking a goal-based approach to business process modeling. This is is based on the notion that a process is a means to change a given business situation with the aim of achieving certain organizational goals. The objective is to reduce or decompose process-related goals until they can be transformed into activities which have to be carried out within the process. Hence a process can be considered a defined set of partially ordered activities intended to reach a goal [123, 217]. It is a means to change a given business situation with the aim of achieving certain organizational goals. Each different situation can be captured and treated as a workflow instance or a case. For example, an order for certain goods placed by a customer creates a business situation that needs to be changed to achieve the goal of making profit. The order fulfillment process consisting of different activities is then a means to achieve this goal.

Also our design approach is based on creating process models for each workflow case instead of a generic workflow process definition or schema. The rationale for the same is as follows. From an organizational point of view, the life cycle of a process includes the phases of planning and coordination. Planning in the development of a process model or definition of what needs to be done in order to achieve the goal. As a simple illustration, creating a ordering of activities in the order fulfillment process

above is a planning task. The coordination phase then involves the execution of this process model by different organizational roles, with respect to different scheduling constraints.

Most existing WFMSs take a process definition as an input and use it to support and automate process coordination. This planning and providing the WFMS with a process definition is done manually. Manual planning is a time and resource intensive task of process specialists, and a costly endeavor as such. Hence, it not economically feasible to plan a process definition for each individual workflow instance. Instead, similar cases are grouped together and planned as a unit, resulting in a generic process definition or workflow schema. For example, there is a single workflow schema for order fulfillment process.

Developing a process definition that captures all variations and copes with all contingencies becomes increasingly difficult as the possible structure of the instances varies. This rigid process structure is hardly capable of reacting in cases of exceptions. This is so because in case of an unanticipated event, the process definition has to be changed and adapted in reaction to the new event, which again involves manual re-planning, in turn delaying the execution. Moreover, the changed workflow schema is now deployed at run time by WFMS, which creates issues for WFMS related to dynamic instance adaptation, endangering the consistency of the process [106, 184, 47, 272, 317]. The underlying reason for these issues is that the WFMS lacks the required information to reason about the dependencies between the different activities in the process steps[1].

Hence, in this work, we propose an integrated planning and coordination approach to process modeling, incorporating automated planning, which has the follow-

[1]Design time dependencies involve pre-conditions and post-conditions for each task, the goal of which is to help reason about the possible ordering between the tasks (design a process). During run time, they get translated into inputs and outputs with specific bindings, for each task. Current process modeling tools use these terms interchangeably (even though no reasoning capabilities are supported), causing semantic confusion.

ing advantages: (1) allows to individually create a process model for every workflow instance, resulting in a better customized and optimized process model, (2) allows incorporation of cybernetic feeback loop through triggering of automatic re-planning in case of process changes and exceptions, thus avoiding process execution delays, and (3) ensuring consistency by capturing all activity dependency information in a domain description, which can be used for reasoning purposes during the planning and re-planning phases.

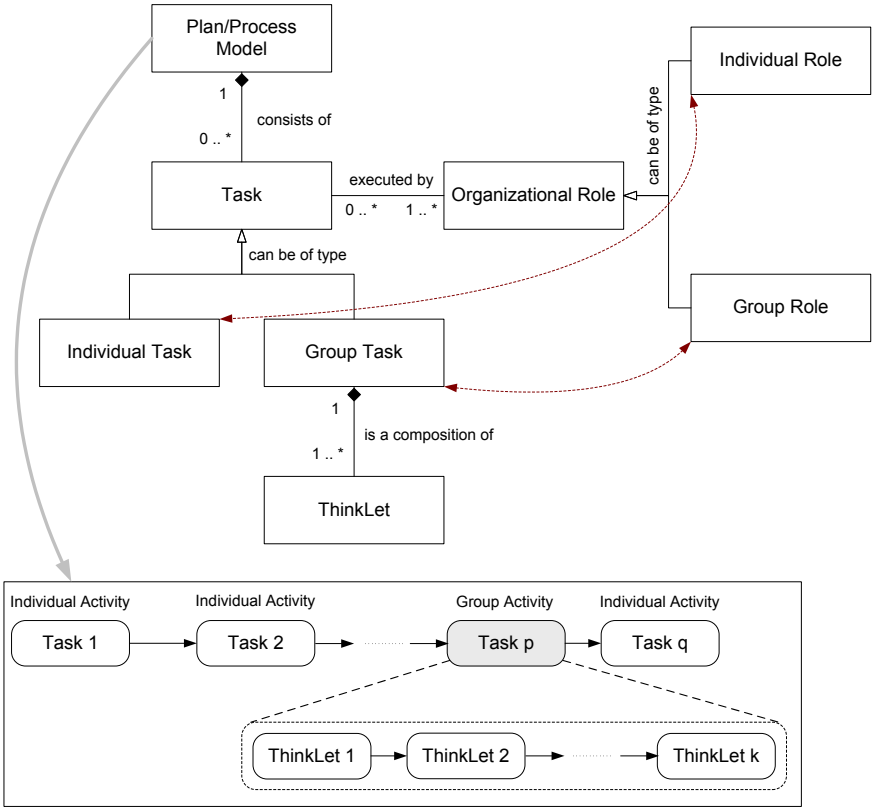


FIGURE 4.1. Static representation of process models

The rationale behind supporting collaborative activities within the overall proposed framework is as follows. Organizational processes involve many intellectual and decision making collaborative activities, such as strategic planning, risk assessment, design review, which are supported using some form of groupware tools. The use of

these tools is mostly restricted to enabling communication and sharing of information among the group members.

Recent research in collaboration has focused on bringing structure to group activities through support of IT tools (e.g., GSS). This can help people work on collaborative tasks in a concerted fashion effectively. One of our goals is to build on this work to propose design approaches that can allow process designers to quickly set up a collaborative task by making use of successful patterns, termed as "thinkLets", observed in previous group sessions.

However, from a holistic process view, it is also necessary to integrate these tools within the overall process management system, so that the information generated during these group sessions can be integrated with other activities for effective process and knowledge management. Hence, another goal of this work is to create process definitions for group activities based on the proposed collaboration process design approach, which can be executed with the help of existing or new group support tools such as GSS.

A static structure of the proposed representation of process models is shown in Figure 4.1. We base our framework on this notion of process models. The figure shows that, an organizational process can be described as a "plan" (in a declarative sense), or equivalently a process model (in an imperative sense). The plan or process model are composed of tasks, each of which could be an individual or group task. Each task is performed by an organizational role, which also could be an individual or group role. The group roles are mapped to the group tasks and the individual roles are mapped to the individual tasks. The process model is also shown as a sequence of interleaved individual and group activities. One such group activity is shown to be composed of collaborative process patterns called "thinkLets".

4.2.2 Structured Process Design Cycle (SPDC)

Using the requirements mentioned in Section 4.1, along with the rationale presented in Section 4.2.1, a Structured Process Design Cycle (SPDC) is proposed. This captures our overall proposed approach to organizational process design, as shown in Figure 4.2. The structured design cycle is shown to be supported by a well-defined knowledge base from which appropriate knowledge is used at different stages of the cycle. The highlighted boxes indicate that the focus of our current work includes steps 1, 2, 3, and 5. Further, the figure also illustrates how each of the steps in the lifecycle is supported. For example, step 3 is currently supported via manual simulation and animation of a process. In the following paragraph, we provide an overview of the various steps.

In steps 1 and 2, process specifications for a given business domain are defined in a well defined ontology (as shown on the right hand side of the figure). These process specifications can be the result of a requirements gathering initiative or a using existing process documentation to restructure the pieces together in comprehensible manner. For a given process, the process specifications may only include the initial conditions, goal conditions of the process and some intermediary constraints. The ontology includes well-defined task descriptions which manipulate the information state of the process (further discussed in Section 5.9). Given such specifications, alternatives process models may be generated using AI planning algorithms. Each process alternative may consist of different task combinations.

In step 3, a process alternative may be selected, possibly based on results of interactive simulation. Different process optimization metrics can be utilized to judge the alternatives. Currently, we support this step by choosing a process alternative with the minimal cost associated. Every task in the process is predefined with cost metrics. Also, we choose to randomly select a process model with similar costs. We believe that this is a naive approach and needs to be refined with process alternative

selection heuristics. Also, different optimization metrics need to be compared. We hope to research these aspects in the future.

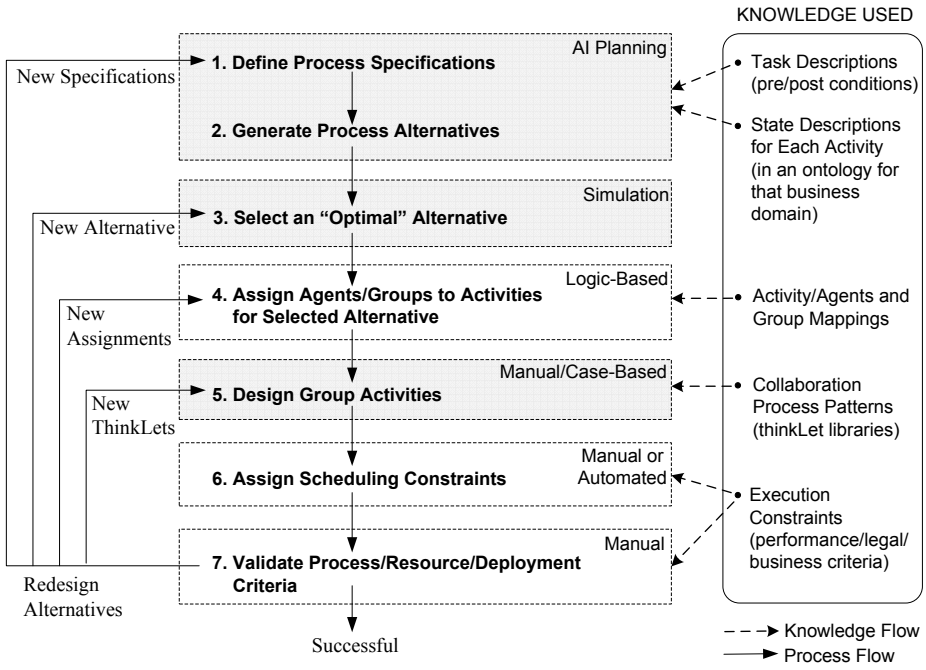


FIGURE 4.2. Proposed Structured Process Design Cycle (SPDC)

In step 4, for each task in the chosen process alternative, an individual agent or a group of agents (based on the organizational chart) may be assigned to execute all the roles associated with that task. Individual tasks can be accomplished usually by a particular organizational member who provides the requisite skills. Group tasks require interaction between and knowledge from different areas of expertise to successfully accomplish the task. An individual may not provide the most effective execution of the task. Our ongoing research is focused on supporting this step with a Description Logic (DL) framework for capturing static relations between tasks and organizational roles [29, 143]. This research would be pursued further to develop design tools for this step.

Group tasks (involving multiple individuals) usually needs to be organized further to allow for efficient information flow and successful meetings [109]. Therefore in

step 5, each group activity is further decomposed into basic collaborative process patterns. The design of collaborative activities based on these collaborative patterns is presented in Chapter 6. In a nutshell, the result of designing a group activity in this manner is a process definition containing an ordering of these patterns, which capture the group interaction along with domain independent and domain specific information, and can potentially be executed with the help of group coordination tools.

Finally in step 6, specific organizational roles are filled with appropriate individuals, resources assigned and tasks are scheduled appropriately. A staged approach to sequencing, resource allocation and scheduling is proposed to reduce the complexity of process design.

Validation of the generated process model may be performed in step 7, based on different criteria. Some of these criteria are shown in the figure.

A feedback cycle is introduced to revise unsuccessful process models appropriately at many stages of the cycle as shown in the figure. This is similar in theme with the systems and control theory, pointed out in Chapter 2.

While each of the stages of SPDC framework are critical to the overall design process, the focus of this dissertation has been on primarily developing appropriate design time tools for supporting stages 1, 2, 3 and 5 of the framework. Toward this end, we have utilized a declarative process formalism to model processes at the task-level of granularity focusing on the information dependencies between tasks and an explicit state representation. Further, to support design of group tasks based on collaborative process patterns, we have developed a formal representation of these patterns, and created a design tool to query these patterns for process knowledge reuse. We describe the formalisms to support steps 1 and 2 in Chapter 5, and step 3 in Chapter 6. We plan to focus our future work on stages 4, 6 and 7 of the framework.

4.2.3 Assumptions

In developing a prototypical system implementing the SPDC framework certain assumptions are made. (1) The optimization metric for generating optimal process model alternatives is assumed to be based on costs allocated to different tasks in the application domain. Other alternatives need to explored in future. (2) Allocation of different roles for performing different tasks is not considered in the implementation. This matching is proposed to be done in future using description logic formalism. (3) Scheduling of resources, including time, is not considered in the implemented prototype. Scheduling is assumed to be a subsequent stage to sequencing. However, concurrency based on ordering constraints is considered during the sequencing stage. (4) The design of collaboration processes, involving concerted collaboration on intellectual tasks, is assumed to be based on existing collaboration patterns (thinkLets). New models need to be developed for other types of collaboration processes along with investigating new patterns.

4.3 Application Examples

In order to illustrate and validate the ideas in the proposed SPDC framework (refer Figure 4.2), we have instantiated the framework for different application domains including (1) loan processing [43, 44], (2) new drug discovery [26, 48, 66, 367], and (3) new product development [315, 280, 279, 371]. We explain the loan approval process and drug discovery process in the following sections. We explain some of the concepts in the later chapters based on these examples. Other examples are also used in exemplifying some concepts and are briefly explained as and when needed.

4.3.1 Evaluation Case: Loan Processing Example

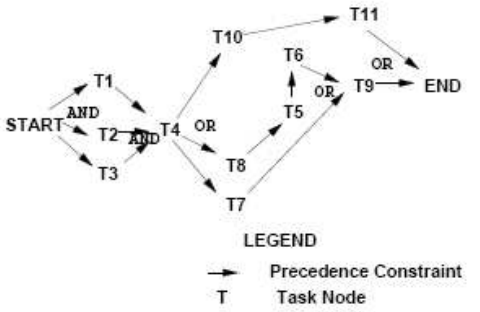
The loan processing example is a structured workflow process that determines whether an application for a property loan is to be accepted or rejected. We borrow this example from Basu and Blanning [43, 44]. The different tasks in this workflow process are listed in Table 4.1. The different information elements (input and output) for each task are also listed in the table. Also, the different organizational roles that may execute these tasks are mentioned in the table (same labeling scheme as in [43] is used to facilitate easy discussion). Figure 4.3 shows the different process model representations for this example.

TABLE 4.1. Summary of tasks in the loan processing example [43]

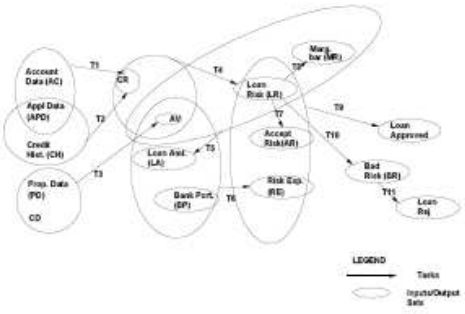
Task ID	**Task Semantics**	**Input Data**	**Output Data**	**Role/Agent**
T1	Calculate Credit Rating	Account Data (AC), Applicant Data (APD)	Credit Rating (CR)	Branch Manager (b)
T2	Calculate Credit Rating	Applicant Data (APD), Credit History (CH)	Credit Rating (CR)	Loan Officer (l)
T3	Appraise Property Value	Property Data (PD), Comparison Data (CD)	Property Value (AV)	Property Appraiser (a)
T4	Calculate Risk Level	Credit Rating (CR), Property Value (AV), Loan Amount (LA)	Loan Risk (LR)	Loan Officer (l)
T5	Revise Loan Amount	Property Value, Loan Risk (LR), Current Loan Amount (LA)	Revised Loan Amount (LA)	Branch Manager (b)
T6	Calculate Risk Exposure	Property Value (AV), Loan Amount (LA), Bank Portfolio (BP)	Risk Exposure (RE)	Risk Analyst (r), Property Appraiser (a)
T7	Evaluate Risk Acceptability	Loan Risk (LR)	Risk Status (AR)	System (s)
T8	Evaluate Risk Marginality	Loan Risk (LR)	Risk Status (MR)	System (s)
T9	Loan Approval	Risk Status (AR)	Loan Status	Loan Officer (l)
T10	Evaluate Risk Unacceptability	Loan Risk (LR)	Risk Status (BR)	System (s)
T11	Loan Rejection	Risk Status (MR)	Loan Status	Loan Officer (l)

4.3.2 Evaluation Case: New Drug Discovery Example

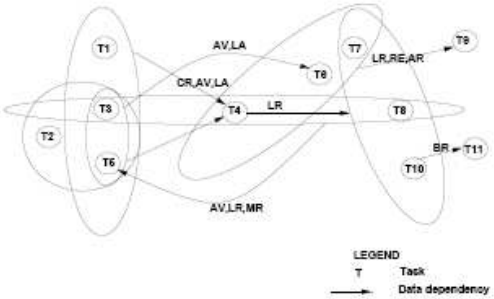
The processes in this domain have been documented during a project conducted at a global pharmaceutical company. This domain is representative of innovative



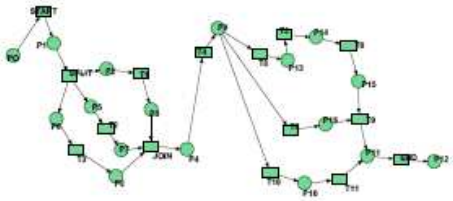
(a) Project Network Representation



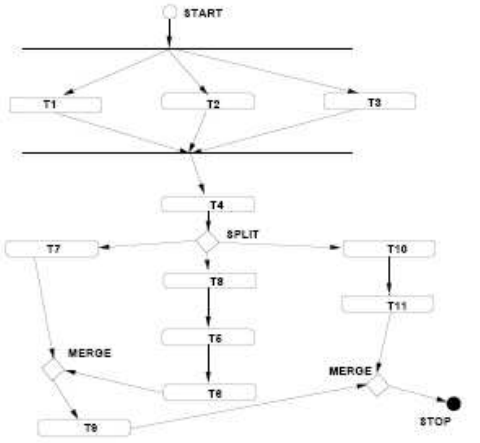
(b) Metagraph Representation



(c) Task Interaction Metagraph



(d) Petri Net model



(e) UML Activity diagram

```
.....Other tags...
<\Activity>
        <Activity Id="HLP_PKG01_Wor1_Act14" Name="Revise Loan Amount">
          <Description>If the risk of the loan is determined to be a
                marginally bad risk (BR), the branch manager uses
                the appraised value of the property (AV) and the level of risk
                associated with the loan (LR) to calculate a new loan amount
          (LA)</Description>
          <Implementation>
            <Tool Id="ReviseLoanAmount" Type="APPLICATION">
              <ActualParameters>
                <ActualParameter>BR_actual</ActualParameter>
                <ActualParameter>LA_actual</ActualParameter>
                <ActualParameter>AV_actual</ActualParameter>
                <ActualParameter>LR_actual</ActualParameter>
              </ActualParameters>
            </Tool>
          </Implementation>
          <Performer>HLP_PKG01_Par1</Performer>
          <StartMode>
            <Automatic/>
          </StartMode>
          <FinishMode>
.....more activities....
```

(f) XPDL Format

FIGURE 4.3. Representations of a loan processing workflow

and unstructured workflow processes. We first outline the domain. At the end, we document some of the challenges in this domain to get some insight into the process.

Drug discovery research consists of the following two different but complementary processes:

- Lead discovery process, and
- Library design and development process

Lead Discovery Process : The goal of lead discovery research is finding and producing chemical compounds that can be used to fight diseases. The viability of chemical compounds as disease-fighting drugs is determined by a set of complex characteristics including potency, toxicity, and so forth.

In most pharmaceutical organizations the lead discovery process contains the following steps [48]. Each of these processes are complex and contain many subprocesses and activities.

1. *Target identification*: A suitable biological target associated with a disease is identified.
2. *Target validation*: The target is validated by conducting several preliminary experiments.
3. *Assay development*: The components to be analyzed are identified; a protocol to sustain the screening of large numbers of compounds against a biological target is developed.
4. *HTS*: Compounds are screened for potency against the biological target.
5. *Hit identification*: A reduced set of chemical compounds, typically of the order of one million, are identified in the HTS phase as inhibitors of the target's biological function.

6. *Lead candidate selection*: The selected set of compounds is further reduced by focusing on the most promising compounds (typical size of this set is 1000 compounds).

7. *Lead optimization*: The chemical structure and biological effects of the selected set are further used to reduce the number of compounds (typical size of the set is 10 compounds).

8. *Preclinical testing*: Selected compounds are pretested before the large scale clinical trials (e.g., testing on animals).

As an example, the assay development process is shown in Figure 4.4

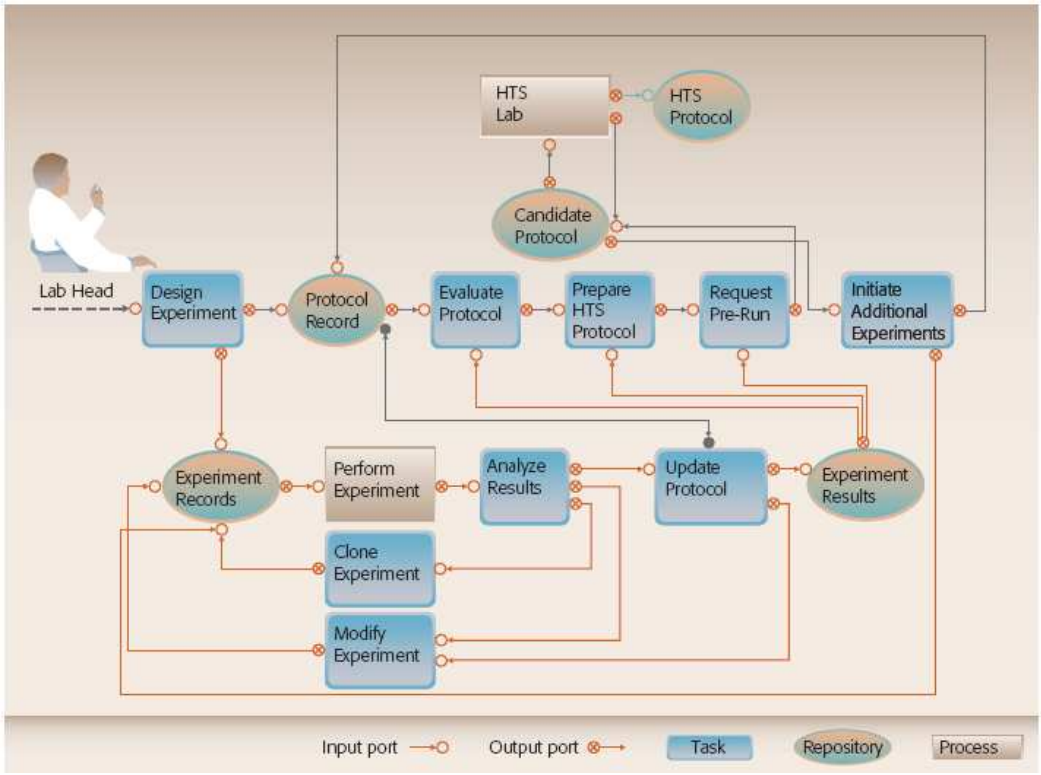


FIGURE 4.4. Assay development process (refer Bhattacharya et al. [48])

Library Design and Development Process : The library design and development process is complementary and parallel process to the main lead discovery process. The

aim of library development is to generate libraries of compounds that can potentially be useful for new drug discovery when a new disease target is to be handled. It is thus a continual process of populating the knowledge base of the organization about the compounds. Similar to the lead discovery process, it has several phases. These phases are shown in Figure 4.5

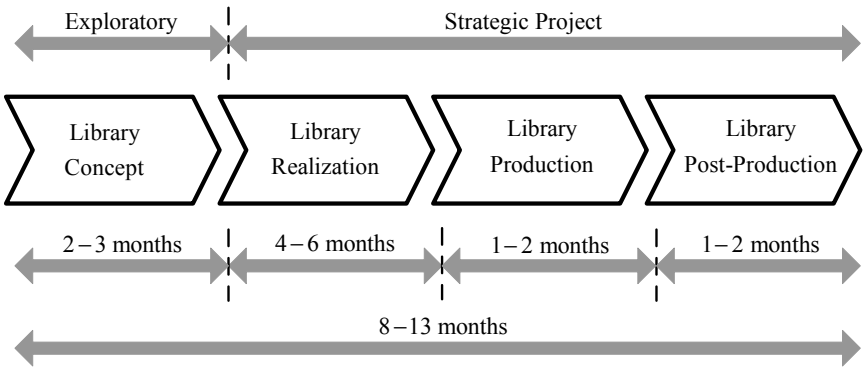


FIGURE 4.5. Overview of the library design process

Each of these phases can be further detailed. As an example, the library concept phase is detailed in the UML activity diagram shown in Figure 4.6. The activity diagram is chosen to show the interaction of different roles in executing the different steps in the process. It is indicated that evaluation of proposals is a collaboration task that involves different team members at different stages in the process.

Challenges in the Drug Discovery Process : As described earlier, the pharmaceutical industry currently faces the challenge of streamlining the drug discovery process. Although cost reduction is one aspect of this goal, the main goal is to produce promising lead compounds as early in the process as possible. We now describe some of the difficulties that the pharmaceutical industry has to cope with in order to achieve this goal [352], which emphasize the need for efficient process management approaches.

- Drug discovery usually involves multiple teams that are dispersed throughout the world. For example, an HTS unit takes over the biological target assays

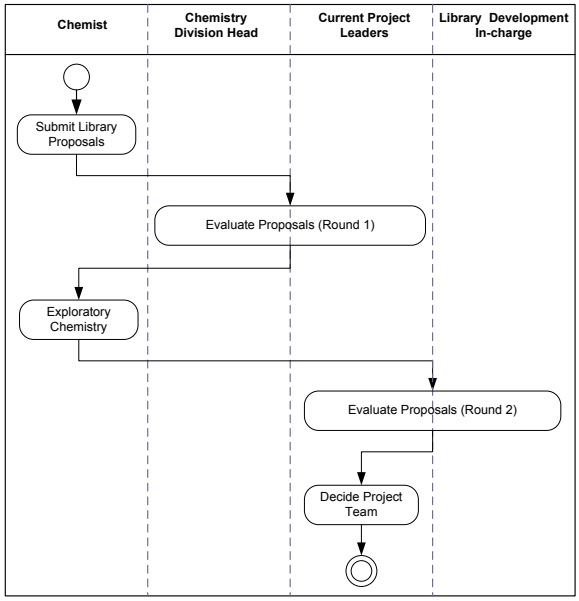


FIGURE 4.6. UML activity diagram for library concept development phase

from therapeutic area teams and screens a large number of chemical structures supplied by chemistry units. The smooth operation of the process demands efficient collaboration among all parties concerned. This collaboration includes the coordination of resources and the exchange of information.

- Typically, every large pharmaceutical research organization has a heterogeneous technology infrastructure. In many instances companies have invested heavily in cutting-edge technology without due consideration for its overall impact on the entire discovery process. Research units often are equipped with platforms that are selected based on local considerations and may not be designed to interoperate with other teams equipment.

- Pharmaceutical research produces a tremendous amount of data, sometimes too vast to analyze in detail. Often data are used by different teams in ways that impede collaboration and the smooth flow of information. For example, the heterogeneous nature of the technology infrastructure leads to information

produced in a variety of data formats. Also, data are typically managed locally within research units and thus not easily accessible by other teams.

4.4 Conclusion

This chapter has presented the overall structured approach to organizational process design, along with two application domains. Next chapters focus on the computational representations by which we can support certain steps of the framework to address the research questions in this dissertation.

Chapter 5

Hierarchical Task Network Planning for Workflow Modeling

In Chapter 2, we looked at WFMS as an enabling technology for achieving coordination between different activities of a business process. WFMS rely on process definitions or process models and use it to facilitate coordination and integration. The generation of suitable process definitions lies at the core of designing organizational process. In the conventional approaches discussed, it is seen that this generation or planning of process models is done manually by the process designer. Manual planning is a resource and time intensive endeavor and thus a costly phase in the deployment of WFMS. In case of manual process planning, it is not economically feasible to plan for every single workflow instance on a case-by-case basis. Hence a well-adopted approach is to group different workflow instances together in a generic process model, called the workflow schema. The manual planning thus involves a process designer conjointly considering similar potential cases, drawing similarities and patterns, and coming up with a generic workflow schema. It is needed to account for peculiarities of workflow instances, by anticipating possible variations of the cases, and modeling alternative execution paths. Often times, this cannot be done completely, because all possible workflow instances are not known a priori. Also, a single process model may be used to coordinate a process, but if an unanticipated (exogenous) event occurs, it becomes necessary to react by modifying the process model. re-planning has to be done manually, and thus delays execution. Following this, the process support system (WFMS), has to adapt itself to the modified process definition. This run-time adaptation of process models has been one of the research focus in this area, as noted in the literature. Since the planning (manual) and execution (coordination by WFMS) are conventionally considered as independent activities, it is difficult for the WFMS

to ensure process consistency, because it does not have all the required information related to the causal dependencies in the process model.

Our goal in this chapter is to investigate these problems and propose a process modeling approach that supports automated process model generation and is flexible to cope with the dynamics of the environment, in addition to the core features of existing process modeling formalisms. In this regard, we begin by drawing an analogy between dynamical systems and business process modeling. Next, applicability of action-state formalisms, specifically situation calculus, is discussed. Following this, AI planning, specifically HTN planning, is presented as a variant of situation calculus to practically encode domain knowledge for facilitating workflow design. An integrated planning and execution approach to process modeling is demonstrated with system description, involving implementation and evaluation.

5.1 Organizations as a Dynamical System

A systemic perspective of organizations was presented in Section 2.1. In competitive modern day markets, organizations are really situated in a dynamic and evolving environment, where adapting business processes is critical. In this regard, organizations can be viewed as dynamical system. To model and analyze organizational processes from a dynamical system, it is helpful to consider how related disciplines have tackled problems of this nature. Over the past several decades, research in this area has resulted in a slew of formalisms and tools including Petri nets, process algebras, dynamic and temporal logics, finite automata, Markov decision processes, differential equations, influence diagrams, etc. Although each of these formalisms have their advantages, most of these formalisms naturally reflect the reference disciplines and applications they stem from. For example, control systems deals with somewhat different problems than, say, compilers, even though both are really examples of dynamical systems. Given this diversity, several researchers are exerting efforts toward a

unifying "theory of dynamics", one that subsumes many special purpose mechanisms that have been developed in these different disciplines. While this is far from being realized, one such attempts is the logical approach to modeling dynamical systems. During this research, we investigated such formalisms for their fit for organizational process modeling that eventually led to our proposed approach.

5.2 On Logic-Based Approach to Modeling Workflows

A promising field of research in analyzing dynamical systems is that of artificial intelligence. Over the past few decades, a number of researchers in artificial intelligence have been developing mathematical and computational foundations for dynamical systems. Brian Smiths' *Knowledge Representation Hypothesis* lays the methodological foundations of all these approaches [353]:

> Any mechanically embodied intelligent process will be be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge.

Adopting this Knowledge Representation (KR) hypothesis has a number of important consequences, which are applicable to the domain of business process modeling (workflow design) as well [322]. It implies that a natural research approach is to employ mathematical logic as a foundation for the "propositional account of the knowledge that the overall process exhibits" called for in part a) of the hypothesis. Providing a propositional account of business processes for an application domain amounts to giving an abstract specification for that business problem. Even by itself, having a non-procedural specification for an application domain is helpful in

explicating the underlying modeling assumptions being made. But in addition to this, because these are logical specifications, it favors reasoning about the domain description (organizational process descriptions for an application domain) based on the causal laws in that domain, in contrast to other approaches involving explicitly enumerating states or transitions (e.g., in case of simulation). Reasoning involves logical entailment at its heart, which maps to different organizational process behaviors. Thus process behaviors can be viewed as logical consequences of the propositional account of the process descriptions for the application domain. Computationally, determining process behavior amounts to logically deducing how it must behave, given the process description, thus implying that logical deduction plays a central role as a computational component. Logical deduction mechanism can also help in establishing correctness properties for the organizational process model. In cases where deduction mechanisms can be performed efficiently, the derived process models are also executable. This means that, as a side effect of providing a logical specification, we can obtain a simulator/coordinator for the system.

Similar reasoning for process development is essential for autonomous agents (such as robots) that cope with dynamic and uncertain environments. The development of expressive and tractable action-state formalisms to support process reasoning has been pursued in the context of developing such agents [331]. Drawing on the application and development of action-state formalisms in the context of cognitive robotics [322, 54], we studied the use of action-state formalisms for representing and reasoning about processes. In the next section, we discuss the potential of using Situation Calculus, which falls under the category of action-state formalisms, for business process modeling.

5.3 Situation Calculus Representation of Workflow Models

Action-state formalisms consist of a formal language that allows adequate specification of action domains and scenarios, and inference mechanisms that tell us precisely what conclusions can be drawn from these specifications. Action domains refer to any aspect of the world worth formalizing in which the execution of actions plays a central role. Action-state formalisms support two fundamental notions, a) a means for describing states, and b) a means to describe transitions between states. We discuss these in the following two paragraphs.[1]

A state is a snapshot of the world being modeled at an instant of time. State descriptions are composed of propositions in some formal logic (usually first-order or propositional logic). A variety of action-state formalisms from basic propositional logic to full first-order logic (including different variants of temporal and event-based logic) have been defined in the literature for describing dynamic systems [331]. Our current discussion is based on the *situation calculus* formalism, in which we reason with situations, which denote states resulting from executing actions [322]. A *situation* may be abstracted as a state with nonzero duration. Informally, each situation is described by a collection of logical terms consisting of the initial situation, S_0, and all situations that are generated by applying an action to a situation. Thus, each situation description includes its history from the initial situation. Each situation is unique by definition, whereas the same state (with infinitesimal duration, a snapshot) may be traversed multiple times, by a dynamic system at different time points. Logical terms in a situation description are called *fluents*, which are relational and functional predicates, that may vary from situation to situation. The fluents represent properties or in general relationships among entities in a situation for a given domain. Thus, in the situation calculus, the state of the world is expressed in terms of functions and relations (fluents) relativized to a particular situation s, e.g., $F(\vec{x}, s)$.

[1] We use the terminology of action-state formalisms (including situation calculus) here and draw analogy to the terminology typically used in the workflow research during the discussion.

Situation descriptions may be used to model both the internal state of a system and state of the external environment.

The second important notion (in action-state formalisms) is an action. Changes in fluents (between situations) may occur as a consequence of actions. Actions cause state transitions when performed. Action descriptions describe the truth value of fluents in a situation that enables that action for execution (preconditions) and also the truth value of fluents in the ensuing state after action (postconditions or effects). An intuitive assumption underlying action-state formalisms is that actions on execution in a situation, affect just a small fraction of the fluents that describe the current state. Describing the effects of an action thus amounts to specifying which fluents change their truth value when an action is executed. Thus, many actions may be eligible for execution in a given state. Further, actions may have deterministic or nondeterministic effects i.e., transitions from a state may be to just one state or any one of a set of states. Further, the effects may be fully or partially observable (to an external observer or to the system). Actions may be executed either by active entities in the system or the environment (which may or may not be modeled in the situational description).

The formalism discussed above may be used to instantiate the elements of a workflow model, namely, tasks, and the data entities (that change due to tasks) and intermediary execution states (of the abstract machines and the environment). A situational calculus description of a process model may then enable a process designer to:

(a) deduce the outcome of a given sequence of tasks (in a domain) on an initial or intermediary situation,

(b) find a suitable sequence of tasks that achieves the desired effects (in a final situation), and

(c) reason about effects of constraints on the tasks, and data entities in the system

or environment.

We briefly sketch the important elements of such a description for supporting workflow design. Given a business domain, we model the tasks in the domain as action descriptions, for all organizational roles. Situations in the domain are described by fluents that capture the relationship among various data entities in the domain. The definition of the tasks and situations in terms of an appropriate domain ontology is called a *domain description*. The initial state of the workflow is described by predicates in the initial situation (modeling the input data). The goal of the business problem (to be resolved by the process model) is described in terms of the fluents in the final situation (modeling the output data). It should be noted that this description is for a single workflow instance and is called the problem description. The *process model* is a sequence of tasks (actions) that transforms the initial situation to the final situation (for that instance) and is called a *plan*. Situation calculus provides first-order predicate logic based entailment operations that determines the set of conclusions that a scenario (the process) within an action domain allows. The entailment operation reflects the special notions of change and causality inherent in descriptions of dynamics in a domain. Given a rich domain description, the development of workflow models can be quite complex. The role of such domain reasoning, planning process interactions, and plan comprehension tasks in software engineering of complex systems has been discussed in [394, 230, 200]. Proposed use of situation calculus for e-business software agents in an open marketplace environment is presented by Albrecht et al. in [8, 229].

The basic elements of situation calculus language $\mathcal{L}_{sitcalc}$ are given in Appendix A. Using these basic elements, the dynamics of a chosen domain is specified by defining the causal laws in language $\mathcal{L}_{sitcalc}$ along with the foundational axioms of the language. The domain description encoding such dynamics, also called a *basic action theory*, would include a collection of axioms $\mathcal{D}$ of the form:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$$

where,

- Σ: the foundational axioms for situations.
- $\mathcal{D}_{ap}$: set of action precondition axioms.
- $\mathcal{D}_{ss}$: set of successor state axioms for functional and relational fluents.
- $\mathcal{D}_{una}$: set of unique names axioms for actions.
- $\mathcal{D}_{S_0}$: set of first-order sentences that are uniform in S_0, so that S_0 is the only term of sort *situation* mentioned by the sentences of $\mathcal{D}_{S_0}$. $\mathcal{D}_{S_0}$ functions as the initial theory of the world (i.e., the one we start off with, before any actions have been "executed") for solving the workflow problem instance at hand.

The axioms Σ, $\mathcal{D}_{ap}$, $\mathcal{D}_{ss}$, and $\mathcal{D}_{una}$ are explained in Appendix A.

The loan processing example discussed in Chapter 4 can be axiomatized using situation calculus. In such a logical declarative description, the fluents would define the data value for each customer (denoted by *cid*), for a property (denoted *pid*) for a loan application (denoted by *lid*). Each fluent would also have the current state (denoted s) as one of the variables. Actions would be parameterized by the appropriate identifiers depending on the set of entities they operate on. The initial state would define the property, the identifier of the applicant, the loan amount and the status of the process. Each precondition axiom would define the fluents that need to be true in the current state for the action to be applicable. For example, action $T1$ is executable when a credit rating for that customer has not been yet assigned by the branch manager. The successor state axioms would list the effect of applying a task on a fluent in any state s. In this example domain the truth value of fluents once assigned are not reversed by other tasks, hence the SSAs turn out to be simple.

In domains with reversible actions,for example, modeling an human resources (HR) workflow wherein employees can be hired or fired, SSAs may be complex.

In such a declarative model, the tasks may include both *information gathering* tasks and *state-update* tasks. Information gathering tasks do not affect the real-world; however, they provide useful information in preparation for a state-update task. The only state-update task of interest is the decision task regarding loan approval that changes the loan status. In our descriptions, we have modeled the effects of information gathering and state-update tasks explicitly. In many real-world domains, such knowledge or information may not be available prior to enactment, and usually has to be determined during execution, which may then lead to different branches of process behavior (based on information gathered during execution).

5.4 Use of the Logical Workflow Formalism

The logical description of tasks, fluents, states, and data entities in a domain along with well-defined inference procedures may be used to support workflow design and management tasks. Given a logical encoding of the domain,the following generic types of reasoning tasks may be performed:

- *Temporal projection*: deducing the outcome of a given sequence of actions
- *Planning*: finding a sequence of actions that achieves a desired situation (or set of effects)

Both temporal projection and planning rely on a central computational mechanism in the situation calculus. This is called *regression*, and it forms the basis of many planning procedures and for automated reasoning in the situation calculus. The intuition underlying regression is this: Suppose we want to prove that a logical sentence W is entailed by some domain description (basic action theory). Suppose further that W mentions a relational fluent atom $F\left(\vec{t}, do\left(\alpha, \sigma\right)\right)$, where F's successor

state axiom is $F\left(\vec{x}, do\left(a, s\right)\right) \equiv \Phi_F\left(\left(\vec{x}\right), a, s\right)$. Then it is possible to determine a logically equivalent sentence W' by substituting $\Phi_F\left(\left(\vec{t}\right), \alpha, \sigma\right)$ for $F\left(\vec{t}, do\left(\alpha, \sigma\right)\right)$ in W. After doing so, the fluent atom $F\left(\vec{t}, do\left(\alpha, \sigma\right)\right)$, involving the complex situation term $do\left(\alpha, \sigma\right)$, has been eliminated in favor of $\Phi_F\left(\left(\vec{t}\right), \alpha, \sigma\right)$, and this involves the simpler situation term σ. In this sense, W' is "closer" to the initial situation S_0 than was W. Moreover, this operation can be repeated until the resulting goal formula mentions only the situation term S_0, after which, intuitively, it should be sufficient to establish this resulting goal using only the sentences of the initial state. Regression is a mechanism that repeatedly performs the above reduction starting with a goal W, ultimately obtaining a logically equivalent goal W_0 whose only situation term is S_0. In this discussion, we indicated how regression works by reducing relational fluent atoms in W; there is an analogous way of reducing the functional fluent terms (see [322]).

Regression enables reasoning backwards from a situation to a previous situation. As an alternative approach to regression, it is possible to reason using progression under certain restricted logical domains (such as transaction-centered databases, robotics). Further properties of these logical operators, regression and progression, are discussed in [322]. Logical unification, resolution, and deduction form the underlying mechanisms for supporting reasoning about constraints given in the situation calculus representation of a domain. Using these operators and mechanisms enables constraint-based what-if analysis of workflow models.

We now discuss such reasoning in the context of the loan processing example. Consider a workflow instance of this domain, where a new loan request is received from a customer (initial state). Possible questions can include: (a) given a sequence of actions, will a loan request be successful? (b) suggest a sequence of actions such that the loan request may be successful, (c) what would be effect of adding a new task to the process model on the approval of loan request, (d) what are the alternatives, if

a particular task fails to execute. It can be noted that computing a response to the first question is equivalent to semantically verifying the behavior of a process model, whereas responding to the remaining questions is equivalent to developing a logically grounded process model. In case (b), the information from the initial state can be used, or in case (c), the domain description with updated tasks can be used, or in case (d), the information from the new current state can be used. Answering the first question (affirmatively or otherwise), requires generic theorem proving techniques using the axioms of situation calculus and mathematical induction. Based on the logical descriptions, a plan may be validated using regression. In the remaining questions, deductive planning is applied, where state transitions are computed using deduction. Thus, plan generation is done by logical deduction, i.e., by applying the inference rules of the logic, rather than by state-space or plan-space search. Given a description of the domain as a set of formulas and a goal as a formula, then a solution plan is generated by a proof in the logic. It can be seen that a plan is equivalent to a path in the process model representations of Figure 4.3 and thus, an instantiation of the process model, for a given instance. We next discuss the practical issues related to using situation calculus as a workflow modeling tool in the real-world and propose the use of Hierarchical Task Network (HTN) planning, a variant of situation calculus [131, 38], for workflow modeling.

5.5 Limitations of Situation Calculus Approach for Workflow Modeling

While situation calculus is a very expressive logic for reasoning with dynamic systems in general, there are practical issues in using such a deductive logic-based approach for workflow design. They include:

- high costs and complexity of encoding domain knowledge in terms of the various axioms of situational calculus,

- the logical formalism does not support workflow (plan) development at multiple levels of abstraction (nested workflows), and
- need to embed control strategies to guide the deduction procedure to combat the computational complexities.

It is informative to contrast the deductive planning approach to the classical AI planning approach to note the pros and cons of the two paradigms. In deductive planning, a planning problem is seen as a deduction problem, i.e., as a problem of proving a theorem in a logical framework (e.g., situation calculus). The main difference between classical planning and deductive planning is that in the former each action is specified with a triple of preconditions, positive effects, and negative effects, whereas in the latter actions are specified with logical formulas. In classical planning, state transitions are computed directly by adding positive effects and deleting negative ones. Deductive planning on the other hand, uses deduction for computing state transition. As mentioned in previous section, plan generation is done by logical deduction, i.e., by applying the inference rules of the logic, rather than by state-space or plan-space search. Given a description of the domain as a set of formulas and a goal as a formula, then a solution plan is generated by a proof in the logic.

The consequences are two fold. From the positive side, the main advantage of deductive planning is its expressiveness. In an expressive logic such as situation calculus, many of the restrictive assumptions of classical planning can be released. It allows for a logical specification of the planning task, stating clearly and unambiguously what the modeling assumptions are, while allowing to prove properties of the process model for the given domain. From the negative side, planning-specific search techniques have shown to be more practical than deduction to solve classical planning problems. The main bottleneck in deductive planning is the lack of automatic procedures to generate plans. The reactions to this problem have been along two main lines. One approach is to limit the expressiveness of the logic to special cases for

which there are efficient decision procedures, such as the "planning as satisfiability" techniques. Another approach, is to keep the logic expressive but allow the process designer to write exhaustive control strategies that can reduce the burden of searching for a proof. For example, situation calculus has been extended to a full blown logic programming language GOLOG, with applications in robotics [231]. With this in perspective, we briefly look at the AI planning techniques, and discuss how an enhancement of these planning techniques, based on heuristics, can help bridge the gap between expressiveness and computational feasibility.

5.6 Artificial Intelligence Planning

AI planning involves generating a network of actions or operators (network of tasks in workflow terminology), i.e., a plan (an instance specific process model), that transforms a world state from an initial state to a goal state. An operator is a parameterized function that transforms a given state into a new state if the operator is applicable in the given state. The parameters to the operator provide access to the various entities in the current state and the new values that their current values may need to be changed to, in order to evolve into the new state. More specifically, a plan is composed of (a) a sequence of operators with a temporal ordering constraint, i.e., $O_i < O_j$ implies that operator O_i must occur sometime before operator O_j; and (b) for each parameterized operator O_i, a set of parameter values that can be applied to the current state to transform it into a new state. A simple formulation of the classical planning problem defines three inputs also called the domain description [54]:

- a description of the initial state of the world in some formal language, usually first-order predicate logic and its variants.
- a description of the goal, i.e., the final state of the world in the same formal language as above.

- a description of the possible operators that can be performed.

A domain-independent planner's output is a sequence of operators – a plan – which, when executed in a world satisfying the initial state description, will achieve the goal. This abstract definition defines a class of problems parameterized by the languages used to represent the worlds, goals, and actions. The capabilities of domain-independent planners vary based on the expressiveness of languages used, assumptions of time, deterministic effects, and so forth. Many approaches to solving these planning problems under different contexts have been proposed in the literature (see [393, 193] for more details). Advances in the speed and performance of AI planning algorithms over the past decade have made it feasible to exploit planning techniques to enable various kinds of reasoning in real-world application contexts (even in interactive applications [278]).

Firstly, we provide a brief background on the state-of-art in planning methodology before outlining our rationale. Two generic classes of planning algorithms have been developed in the AI planning literature, namely, partial order planners and total order planners [331]. Partial order planners develop project networks, consisting of directed networks of actions, whereas total order planners develop a plan with sequential actions. A given partial order plan may be linearized into multiple sequential plan alternatives. Partial order planners search in plan space, whereas total order planners search in state space.

A variety of algorithms have been developed for both types of planners, based on techniques such as constraint satisfaction, logic resolution and theorem proving, satisfiability and model checking; wherein each algorithm involves searching by using some combination of *refinement* (modifying the collection of actions and/or constraints associated with a node), *branching* (generating one or more children of the current node, which will be the candidates for the next node to visit), *pruning* (removing from the set of candidate nodes, some nodes that appear to be unpromising for the search),

and *selecting* (nondeterministically choosing among the current node's nonpruned children, if any, to get the next node to visit in the search space) steps, in addition to a termination step [331, 140]. In general the search procedure will perform well if these steps can do a good job of organizing the search space an guiding the search.

Partial order planners embed a least commitment strategy and the plans they generate are inherently flexible because of their networked nature. However, developing such partial order plans incurs higher plan search costs unlike state-space planners which have exhibited excellent performance on large problems [115, 34, 192]. Recent research in AI planning has been focused on developing plans involving concurrent and durative actions for both partial and total order planners. However, developing such plans with parallel actions requires explicit temporal models for actions such as their durations and metric models of resources in the plan representation. Consideration of such numeric quantities considerably increases the size of the search space [31]. Developing efficient and scalable algorithms for temporal and metric reasoning in planning domains is an active research area whose results may be incorporated in our framework in future. An alternative approach used in our framework is twofold, first the generation of totally ordered sequential plans, followed by the relaxation of precedence constraints to generate partially ordered plans [33, 407]. Details are further discussed in Section 5.7.

For purpose of workflow modeling, our choice of a planning approach was guided by the need for (a) performance, (b) the ability to embed domain knowledge to guide and control the search for a viable plan, and (c) an explicit state representation to enable interleaving of planning and execution. These criteria guided our decision use to a state-space, linear-order planning algorithm to implement the process alternative generation step in the conceptual framework (step 2 in Figure 4.2). This also favors the post-processing step of relaxing the precedence constraints to generate partial order plans with concurrent actions.

A number of heuristic techniques have been proposed for improving the efficiency of planning. Sometimes, the heuristics are domain-independent, i.e., intended for use in many different planning domains. The sole input to the planner is a description of a planning problem to solve, and the planning engine must be general enough to work in any planning domain within some large class planning domains D. Historically, D was often assumed to be the set of all "classical" planning domains, a class too restricted to include most practical planning applications. However, automated-planning researchers use this assumption less frequently as they become more interested in extending their work beyond classical planning. On the other end of the spectrum are domain-specific planners, which are tailor-made for a given domain and would not work in other domains without major modifications. This class includes some of the successful planners deployed in practical applications [357]. Domain-configurable planners fall in the middle of the spectrum. In their case, the planning engine is domain independent but the input to the planner includes domain specific control knowledge, that is, information to help the planning engine solve problems in the relevant problem domain. This reduces the effort of the domain encoder by needing to only a language to provide the domain knowledge to the planner. The details of the generic planning are taken care of by the planning engine. These planners have sometimes been called "hand-tailorable" planners. In the recent two AI planning competitions [30, 128], the systems that were fastest, could solve the hardest problems, and could handle the widest variety of planning domains all incorporated domain specific control of the planning process. Furthermore, nearly every successful planning system for real-world applications incorporates domain specific heuristics or even domain specific algorithms.

The ability to use domain-specific problem-solving knowledge can dramatically improve a planner's performance, and sometimes make the difference between solving a problem in exponential time and solving it in polynomial time (see e.g., [161, 351]. In experimental studies (see e.g., [285, 282, 32]), hand-tailorable planners have quickly

solved planning problems orders of magnitude more complicated than those typically solved by "fully automated" planning systems in which the domain-specific knowledge consists only of the planning operators.

With this motivation, we have chosen Hierarchical Task Network (HTN) planning, which is a domain-configurable AI planning methodology that uses task decomposition to encode branching rules. HTN planning is a variant of situational calculus as discussed in [131, 38]. HTN planning provides a formalism to encode the domain axioms in a convenient and incremental manner along with a scalable search algorithm for generating plans. HTN planning has been more widely used for practical applications than any of the other planning techniques [140]. This is partly because HTN methods provide a convenient way to encode problem-solving "recipes" that correspond to how a human domain expert might think about solving a planning problem. This is particularly applicable in the workflow domain where the process designer has to encode the requirements gathered from domain experts and model the business process to enable solving multiple problem instances in that domain. We now delve into HTN planning in detail to discuss how we can leverage this technique for developing support tools for the workflow modeler.

5.7 Hierarchical Task Network (HTN) Planning and SHOP2

5.7.1 HTN Planning

HTN planning was first developed more than 25 years ago [332, 365]. Historically, most of the HTN planning researchers have focused on practical applications. Examples include production-line scheduling [406], crisis management and logistics [75], planning and scheduling for spacecraft [2], equipment configuration [7], manufacturability analysis [174, 356], evacuation planning [274], and the game of bridge [357]. Although HTN planning has been used for process modeling in specific application

domains, its proposed application in the context of general workflow modeling is novel.

The development of a formal semantics for HTN planning has shown that it is strictly more expressive than classical AI planning: there are some problems that can be expressed as HTN planning problems but not as classical planning problems (unlike classical planning, HTN planning is Turing-complete) [113, 114]. Even if one places restrictions on HTN planning to restrict its expressive power to that of classical planning, it generally is much easier to translate classical planning problems into HTN planning problems than vice versa [236].

HTN planning is like classical AI planning in that each state of the world is represented by a set of atoms, and each action corresponds to a deterministic state transition. However, HTN planners differ from classical planners in what they plan for and how they plan for it. In an HTN planner, the objective is not to achieve a set of goals but instead perform some set of *tasks* (symbolic represenations of activities to be performed). The input to the planning system includes a set of *operators* similar to those of classical planning and also a set of *methods*, each of which is a prescription for how to decompose some task into some set of *subtasks* (smaller tasks). Planning proceeds by using methods to decompose *nonprimitive tasks* recursively into smaller and smaller subtasks, until *primitive tasks* are reached that the plan executor can perform directly using the planning operators.

An HTN planning problem consists of the initial state (a symbolic representation of the state of the world at the time the plan executor will begin executing its plan), the initial task network (a set of tasks to be performed, along with some constraints that must be satisfied), and a *domain description* that contains the following:

- A set of planning *operators* describing what actions (that is, what primitive task types) the plan executor can perform. Each operator can have a set of preconditions that must be true before it can be executed and a set of effects

that will occur afterward. An action (or synonymously, a primitive task) is an operator instance, produced by assigning values to an operators parameters.

- A set of *methods* describing possible ways of decomposing tasks into subtasks. These are the "standard operating procedures" normally used to perform tasks in the domain. Each method can have a set of constraints that the world state must satisfy before the planner can apply the method.

- Optional information such as definitions of auxiliary *functions* and of *axioms* for inferring conditions not explicitly mentioned in states of the world.

5.7.2 Ordered Task Decomposition

Ordered task decomposition is a special case of HTN planning in which the planning algorithm always builds plans forward from the initial state of the world[286]. In other words, an ordered-task-decomposition planner plans for tasks in the same order that the tasks will later be performed. The first applications of ordered task decomposition were tailor-made for specific application domains. The best known example is the code for declarer play that helped Bridge Baron win the 1997 world championship of computer bridge [357]. For this research, we have used the Simple Hierarchical Ordered Planner 2 (SHOP2) system [284], which is an HTN planner using ordered task decomposition and constraint satisfaction as its search-control strategy.

There are several ways in which the HTN planning approach is promising for workflow modeling:

- HTN encourages modularity. Methods can be written without consideration of how its subtasks will decompose or what compound tasks it decomposes. The method author is encouraged to focus on the particular level of decomposition at hand.

- This modularity fits in well with workflow modeling. Methods correspond to recursively composable workflows. These workflows can come from diverse independent sources and then integrated by the planner to produce situation specific, instantiated workflows. Thus, it can support nested workflows through multiple levels of abstractions.

- Since the planner considers the entire execution path, it has opportunities to minimize various sorts of failures or costs. Most obviously, if the planner finds a plan, one knows that the top level task is achievable with the information at hand. If the granularity of the workflow tasks is large enough, then it can be considerably easier for a human being to inspect and understand the plan.

- HTN planning scales well to large numbers of methods and operators.

- Some HTN planners (e.g., SHOP2) support complex precondition reasoning, and even the evaluation of arbitrary code at plan time. These features make it straightforward to, integrate existing knowledge bases on the process models as well as the information supplying applications.

- HTN planning provides natural places for human intervention at plan time. The two obvious examples are first, that in preconditions, a code or service call can query a person for special input, and second, if the planner hits a point where it cannot continue decomposition, it can request a decomposition of that step from another person, or even a software agent[2].

- It is often argued by HTN practitioners that such representations are more appropriate for many real-world domains than are classical planning operators, as they better characterize the way that users think about problems [283].

[2]For example, the HiCAP [8] system employed SHOP as a component of a mixed initiative system [273].

In the proposed framework, we utilize the HTN planning technique to support: a) basic workflow generation, b) workflow generation with reuse, c) concurrent plan generation, d) generation of control flows, e) generation of nested plans, f) constraint reasoning and g) interleaving planning and execution. We next describe these applications of HTN planning for workflow modeling below.

5.8 HTN Planning for Organizational Process Design

HTN planning is proposed to be used at two stages. Firstly, for generating of process alternatives by exploring the design space (for a given workflow instance), as shown in step 3 of the overall conceptual framework (Figure 4.2). Secondly, to support the execution of the overall structured design cycle, by interleaving planning and execution, which can lead to more adaptive and robust business process management. Each of these applications are illustrated below. But, first the SHOP2 planner is introduced, which forms the planning engine of the prototype design support tool developed.

5.8.1 SHOP2

SHOP2 [284] is based on SHOP [285], a previous domain-independent HTN ordered task decomposition planner that requires the subtasks of each method, and also the initial set of tasks for the planning problem, to be totally ordered rather than partially ordered. Thus in SHOP, subtasks of different tasks cannot be interleaved. SHOP2 extends SHOP by allowing the subtasks of each method to be partially ordered. Experiments have shown that this can allow SHOP2 to create plans more efficiently than SHOP, using domain descriptions simpler than those needed by SHOP [282]. Both SHOP and SHOP2 have been implemented in LISP and use a lisp-like lambda calculus formalism. Both SHOP and SHOP2 are available as open-source software

at http://www.cs.umd.edu/projects/shop and have been used in various real-world applications as shown in [283].

The important syntactic and semantic elements of the HTN planning approach used in SHOP2, relevant to its use in our process modeling framework are briefly discussed, based on [285, 284, 140].

```
procedure SHOP2(s, T, D)
  P = the empty plan
  T_0 ← {t ∈ T : no other task in T is constrained to precede t}
  loop
    if T = ∅ then return P
    nondeterministically choose any t ∈ T_0
    if t is a primitive task then
      A ← {(a, θ) : a is a ground instance of an operator in D, θ is a substi-
            tution that unifies {head(a), t}, and s satisfies a's preconditions}
      if A = ∅ then return failure
      nondeterministically choose a pair (a, θ) ∈ A
      modify s by deleting del(a) and adding add(a)
      append a to P
      modify T by removing t and applying θ
      T_0 ← {t ∈ T : no task in T is constrained to precede t}
    else
      M ← {(m, θ) : m is an instance of a method in D, θ unifies {head(m), t},
            pre(m) is true in s, and m and θ are as general as possible}
      if M = ∅ then return failure
      nondeterministically choose a pair (m, θ) ∈ M
      modify T by removing t, adding sub(m), constraining each task
        in sub(m) to precede the tasks that t preceded, and applying θ
      if sub(m) ≠ ∅ then
        T_0 ← {t ∈ sub(m) : no task in T is constrained to precede t}
      else T_0 ← {t ∈ T : no task in T is constrained to precede t}
  repeat
end SHOP2
```

FIGURE 5.1. A simplified version of the SHOP2 planning algorithm [284]

SHOP2 uses first-order logic definitions of variable and constant symbols, function and predicate symbols, atoms, conjuncts, most general satisfiers and Horn clauses. A *state* is a set of ground atoms and an axiom set is a set of Horn clauses. If S is a state and X is an axiom set then $S \cup X$ satisfies a conjunct C if there is a substitution u

(called a satisfier) such that $S \cup X$ entails C. u is a most general satisfier (called mgs in short) if there is no other satisfier v more general than u. There may be several distinct mgs's for C from S and X. A *task* is a list of the form $(s\, t_1\, t_2 \ldots t_n)$, where s (the tasks name) is a task symbol and t_i (the task arguments) are terms. The task is primitive if s is a primitive task symbol (denoted $!s$) and it is compound otherwise. A task list is a list of tasks. An *operator* is an expression $(\texttt{:operator}\, h\, P\, D\, A\, [c])$ where h (the head) is a primitive task, P is a list of preconditions (expressed as a logical expression), D and A (the deletions and additions) are sets of atoms containing no variable symbols than those in h, and the optional cost expression c for the operator (the default value is 1). For example, a primitive operator in the workflow modeling context is a operator corresponding to a task (individual or group, and automated or manual) that would be performed by the appropriate organizational role(s) or a software tool. Primitive operators manipulate the state description of the planning domain. A *method* is an expression of the form $(\texttt{:method}\, h\, C_1\, T_1\, C_2\, T_2 \ldots C_k\, T_k)$, where h (the head) is a compound task, C_i is the method's precondition expressed as a conjunct and T_i (the tail) is a task list.

The intent of an operator $o = (\texttt{:operator}\, h\, P\, D\, A\, [c])$ is to specify that h can be accomplished by modifying the current state of the world by removing every atom in D and adding every atom in A. More specifically, if t is a primitive task and there is an mgu (most general unifier) u for t and h such that h^u is ground, then o is applicable, and that h^u is a simple plan for t. If S is some state, then the state produced by executing o^u (or equivalently h^u) in S and L is the new state $h^u(S) = o^u(S) = (S - D^u) \cup A^u$.

The intent of a method $m = (\texttt{:method}\, h\, C_1\, T_1\, C_2\, T_2 \ldots C_k\, T_k)$ is to specify that if the current state of the world satisfies C_i, then h can be accomplished by performing the tasks in T_i with the specified ordering. More specifically, let S be a state, X be a set of axioms and t be a task. Suppose there is an mgs u that unifies t with h and suppose $S \cup X$ satisfies ${C_i}^u$, then m is applicable to t in $S \cup X$ and the result

of applying m to t is the set of task lists $R = \{(T_i^u)^v$: v is an mgs for C_i^u from $S\}$. Each task list r in R is a *simple reduction* of t by m in $S \cup X$.

A *plan* is a list of heads (in the lambda calculus sense) of ground operator instances. If p is a plan and S is a state, then $p(S)$ is the state produced by starting with S and executing the operator instances in the given order. A business process can be expressed as a *planning problem*, which is a tuple $P = (S, T, D)$, where S is the initial state (consisting of a priori available information), T is a task list (consisting of the tasks to be accomplished to achieve a business process goal) and D is a domain description consisting of a set of operators, axioms and methods (the task descriptions for the business process in context). Given the business process as a planning problem (S, T, D), $\Pi(S, T, D)$ is the set of all plans (process models) for T from S in D, which is defined recursively as follows.

If T is empty, then $\Pi(S, T, D)$ contains exactly one plan, namely the empty plan. Otherwise let t be the first task in T and R be remaining tasks. There are three cases: (1) If t is primitive and there is a simple plan p for t, then $\Pi(S, T, D) = \{append(p, q) : q \in \Pi(p(S), R, D)\}$. (2) If t is primitive and there is no simple plan for t, then $\Pi(S, T, D) = \emptyset$. (3) If t is compound, then $\Pi(S, T, D) = \cup\{\Pi(S, append(r, R), D)$: r is a simple reduction of $t\}$. A simplified version of the SHOP2 algorithm for computing $\Pi(S, T, D)$, based on the above definitions, is given in Figure 5.1.

The SHOP2 algorithm is shown to be complete and sound in [285]. State spaces for planning problems can be finite or infinite depending on the underlying domain description. The SHOP2 algorithm is recursive in nature and termination criteria include stopping after finding a single plan, exhaustively exploring the search space to find all plans and then find an optimal plan. Each such criteria would incur different search costs, providing opportunities for optimization. For our purposes, we treat the SHOP2 planner as a black box and focus on its use for process modeling by interlacing planning and execution. Given task descriptions for a business process

domain, this planner computes multiple plans (process models) for a given business process problem.

5.9 Using AI Planning for Workflow Design

Referring to the SPDC framework in Figure 4.2, we propose the use of AI planning for supporting steps 1, 2, and 3, along with the feedback loop for these steps. The following steps illustrate the use of planning in creating a process model for a single workflow instance.

(a) Develop a domain ontology based on first order logic [SPDC step 1]

(b) Develop a representation for tasks and states in terms of the domain ontology (see Section 5.7.1) [SPDC step 1]

(c) Define the initial states and goal states for the workflow instance at hand [SPDC step 1]

(d) Use the SHOP2 planner to generate alternative plans (process models) [SPDC step 2]

(e) Select the optimal alternative based on optimization metrics, such as activity costs in SHOP2 [SPDC step 3]

(f) Interleave planning and execution for reactivity and flexibility [SPDC steps 1–2, 3 loop]

A planning-based approach to process modeling supports generality in the following steps: (1) The same planning algorithm may be used in multiple problem domains, for example, loan processing, online shopping, new drug discovery, and so forth (as discussed in Section 5.11.1. This would require the definition of domain ontologies (with different task descriptions) for each appropriate domain (or processes

in the organization. The generality is supported by the use of first order logic for reasoning. However, process knowledge reuse can be facilitated by using related task or state descriptions from different processes or application domains. (2) Once a domain is defined, different types of problems may be posed in the domain by choice of appropriate state predicates defining the initial and final states (thus eliminating steps (a) and (b) mentioned above), and (3) for each type of problem, multiple ground instances may be defined. Interleaving planning and execution lies at the core of this proposed approach, which is explained next.

5.9.1 Interleaving Planning and Execution

Since the process definition using automated planning is generated for every case, the quality and flexibility can be improved over traditional process modeling ways implementing a schema design approach [68, 242, 327, 340, 417]. An important aspect of flexibility is the ability to react to unanticipated events that might occur during execution, and that of quality is to gather and use accurate and timely information for improved planning.

Flexibility in process design can be introduced by re-planning and dynamically adapting the process. Three important steps can be noted in re-planning of process definitions (whether or not automated planning is used) [341]:

1. Identifying when re-planning is needed
2. Generating a new process definition
3. Adapt process enactment to the changed process definition

Most of the current research focus in the workflow area, specifically flexible workflows, has focused on the dynamic adaptation of workflows (step 3 above), i.e., adjusting executing instances to changes in the process definition [106, 184, 47, 272, 317].

Recently, some research approaches have put forth the idea of planning process definitions automatically to improve quality and flexibility [132, 341]. In this work, we focus on supporting all the three steps noted above with an integrated planning and enactment system discussed below.

Embedding flexibility necessitates an interlacing between the planning and execution phases of process modeling. It is well-recognized complex problem to decide when and how to interleave execution and planning [18, 259, 270]. This issue has attracted considerable attention in the context of building complex and robust autonomous control systems for a diverse set of applications [228, 291, 392]. Recently, combined planning and execution has been studied for the problem of automated web service composition [241].

Also, various interleaving strategies have been proposed by researchers to deal with execution failures. They typically consist of re-planning, reactive planning or plain re-execution. re-planning on failure involves developing a new plan considering the latest world information and then executing the newly generated plan. Reactive planning on failure involves modifying the current plan according to rules that define how the plan should be tweaked for each possible kind of failure. Re-execution on failure strategies attempt to re-execute the initial plan with the belief that the root causes of plan failure are random and the world will return to a state that is consistent with the assumptions of the plan. Thus if the world returns to a consistent state with respect to initial assumptions, the initial plan will succeed. Each of these strategies assumes different kinds of knowledge about changes to the world from the time the plan was conceived [331]. Here we build on these works to propose a dynamic approach to process modeling with the re-planning strategy.

In real-world organizational settings, there can be many ways in which things can go wrong during a process. We identify some of them here. Firstly, as noted above, execution of an activity may fail. For example, the task operators can time out due to network or source dynamics. Secondly, a task may be added, removed

or changed in the domain description, rendering the old process plan outdated. For example, a credit check may no longer be needed for orders below certain amount, which will change the task operator's functionality. Thirdly, an external (exogenous) event may cause unanticipated effects to be added to the process state. For example, additional information may become available about the case during the enactment of the process. Fourthly, the goal of the process itself may change during the execution. For example, a customer may call and change his order. Each of these cases require to trigger re-planning in order to generate new models that are workable as well as optimal [341].

Along with reacting to dynamic world changes, another important reason for marrying planning with execution is information gathering. As noted in Section 5.3, tasks can be classified as either *state-update* tasks or *information gathering* tasks. In classical AI planning, given a planning problem for a domain, a planner is designed to return a sequence of actions to be executed later. However, this assumes that all the knowledge to create a plan is available a priori. In some cases, information gathering operators are executed during planning to gather relevant information from the environment to proceed with the planning [219]. However, this is not always possible in most real-world business processes. This is because many tasks are decision making tasks, whose effects cannot be known before the planner kicks off. In such cases, information gathering via real execution can make the planner proceed in a more informed manner. Also, the chances of needing to re-plan can be expected to decrease during the execution phase.

While using execution as a source of information is an interesting concept, it is often confounded with many underlying issues. As planning proceeds incrementally, new plan steps are introduced into the plan one by one, making choices and commitments. For completeness of the planning algorithm, all the choices must have a fair chance of being included, thus implying that the planning search space should not be pruned to eliminate a potential solution, due to a committment made during

the planning process. Hence, every choice during planning is reversible through backtracking, leading to exploration of different plan refinements to achieve the goal state. In this sense, real execution of an action during the planning process removes some control from the planning algorithm, but disabling backtracking over past committments. On the other hand, real-world execution functions as real-world sensing by providing additional knowledge to for the planning process. These issues are detailed by Stone and Veloso in [361].

Advancements in AI planning research are helpful in tackling this issue. The SHOP2 algorithm chosen is based on the search-control strategy of ordered task decomposition, as mentioned earlier. This strategy break tasks into subtask and generates the plan's actions in the same order that the plan executor will execute them. So, throughout the planning process, the planner can tell what the state of the world will be at each step of the plan. This reduces the complexity of reasoning by reducing a great deal of uncertainty about the world, thereby making it easy to incorporate substantial expressive power into the planning system.

5.9.2 Developing Concurrent Plans

The output of the SHOP2 algorithm is a sequential totally ordered plan. One possibility to generate a plan with concurrent actions is to post-process this plan based on a greedy variant of the Minimal Deordering algorithm (based on [33, 241]) called PlanParallelize. The algorithm is shown in Figure 5.2.

Given a sequential plan, and the current state, the PlanParallelize algorithm selects all the applicable actions to the current state, while checking that no two actions conflict in the current state, based on the precedence constraints. It updates the current state with the set of applicable actions to generate a successor state and repeats the process on the remaining set of actions (that were not applicable in the previous state). The algorithm terminates when there are no more actions to parallelize. An

procedure PlanParallelize
Input: (1) A valid totally ordered plan $P = \langle A, \prec \rangle$,
where $A = \langle a_1, a_2, \ldots, a_z \rangle$ is an action sequence,
$\prec$ is the set of precedence constraints between these actions,
(2) the initial state, s_{init}
Output: A deordering of P, A_c consisting of sets of actions
$\big((a_2, \ldots, a_n), (a_{n+1}, a_{n+2}, \ldots, a_m), \ldots, (a_m, \ldots, a_z)\big)$.
A_c is the set of partially ordered actions.
(1) Initialize variables: $s_{curr} = s_{init}$, $A_c = \varnothing$
(2) If $A = \varnothing$, stop.
(3) Else, for $a \in A$, if a is applicable to s_{curr} without violating constraints in $\prec$,
collect a into *templist*. Apply actions a in *templist* to s_{curr} to update s_{curr}.
Append *templist* to A_c. Update $A \leftarrow (A - templist)$. Go to step 2.
(4) Finally return A_c.
end PlanParallelize

FIGURE 5.2. PlanParallelize: A minimal deordering algorithm for generating concurrent plans [33, 241]

example of this shown in later sections.

5.9.3 Modeling of Different Control Flows with SHOP2

Various control flow constructs can be embedded in SHOP2 domain descriptions by ways of defining methods and operators, thus guiding the search through the process design state space. Some of the control flow constructs typically used in the context of workflow modeling are described here.

Sequence : This construct can be realized by defining a method m, as m = (`:method` $h\ C\ T$), where T = *ordered* task list of the form $(s\ t_1\ t_2 \ldots t_n)$.

If-Then-Else : This construct can be realized by defining a method m, as m = (`:method` $h\ C_1\ T_1\ C_2\ T_2$), where C_1 = conjunct of all preconditions for the method and conditions for *If* (denoted as π_{if}), T_1 = task list of the form $(s\ t_1\ t_2 \ldots t_n)$ for

Then, C_2 = conjunct of all preconditions for the method, and T_2 = task list of the form $(s\ t_1\ t_2 \ldots t_n)$ for *Else*.

Repeat-While : This construct can be realized by defining a collection of methods (m_1, m_2), such that
$m_1 = (\texttt{:method}\ h_1\ C_1\ T_1)$, where C_1 = conjunct of all preconditions for the method, and T_1 = task list of the form $(s\ t_1)$ with $t_1 = h_2$ of method m_2; and
$m_2 = (\texttt{:method}\ h_2\ C_1\ T_1\ C_2\ T_2)$, where C_1 = conditions for *While* (denoted as π_{while}), T_1 = task list of the form $(s\ t_1\ t_2 \ldots t_n)$ with $t_n = h_2$, $C_2 = \emptyset$, $T_2 = \emptyset$.

Repeat-Until : This construct can be realized by defining a collection of methods (m_1, m_2), such that
$m_1 = (\texttt{:method}\ h_1\ C_1\ T_1)$, where C_1 = conjunct of all preconditions for the method, and T_1 = task list of the form $(s\ t_1)$ with $t_1 = h_2$ of method m_2; and
$m_2 = (\texttt{:method}\ h_2\ C_1\ T_1\ C_2\ T_2)$, where C_1 = negation of conditions for *Until* (denoted as (not π_{until})), T_1 = task list of the form $(s\ t_1\ t_2 \ldots t_n)$ with $t_n = h_2$, $C_2 = \emptyset$, $T_2 = \emptyset$.

Choice : This construct of a choice between tasks $t_1, \ldots, t_n$ can be realized by defining a collection of methods $(m_1, \ldots, m_n)$, such that $m_i = (\texttt{:method}\ h_i\ C_i\ T_i)$, where C_i = conjunct of all preconditions for the method, and T_i = task list of the form $(s\ t_1\ t_2 \ldots t_n)$. This construct is a key construct in generation of process model alternatives.

Unordered : This construct of designing tasks $t_1, \ldots, t_n$ without any ordering constraints between them, can be realized by defining a method m, such that $m = (\texttt{:method}\ h\ C\ T)$, where C_i = conjunct of all preconditions for the method, and T = *unordered* task list of the form $(s\ t_1\ t_2 \ldots t_n)$. This construct is a key construct in generation of partial orderings between tasks.

5.10 Prototype System

Based on the observations in the above section, our prototype system implements integrated planning and execution components that can allow information gathering during execution (based on information generated during the process itself), as well as re-planning in case of unexpected failures of tasks or any exogenous events.

The general architecture of the system is shown in Figure 5.3. The prototype system consists of four main components, the planning module, the controller module, the collaboration process module, and the world monitor. For the planning module, SHOP2 planner is utilized and run as Lisp servlet, while the controller module is developed as Java servlet. The planning and collaboration process modules interact with the controller module. The world monitor runs concurrently with other modules and monitors state changes.

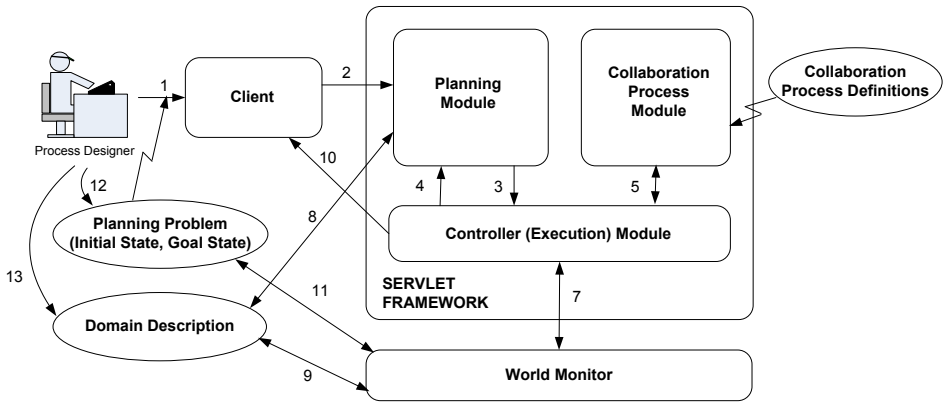


FIGURE 5.3. Prototype system architecture

The generic steps of how the system functions are indicated by the numbers on arrows. A goal-oriented workflow instance as well as the current state, modeled as a planning problem are provided to the planning module from a browser interface (steps 1 and 2). The business process requirements and constraints, encoded as a planning domain description, are made available to the planning module (step 8). Based on this information, the planner generates possible process models (plans). Plans

could consist of: (1) feasible alternative plans with different tasks and bindings, (2) feasible alternative plans with same tasks, but different data bindings, or (3) feasible alternative plans with same tasks and bindings, with different partial orderings.

The generated set of plans is passed on to the controller module (step 3). Optimal alternative is then selected based on cost metrics (provided for tasks in the domain description). Currently, we opt for a simple minimal cost strategy. If different plans have the same cost, a plan is randomly chosen from that set for execution. This step needs further refinement in terms of other optimization possibilities and possible ways of making choices between different plans, which is one of future extensions planned. In instances with incomplete information, complete plan generation is not possible during the planning phase. In such cases, a strategy of choosing the first plan (generated so far) with the least cost is incorporated.

The controller passes the selected plan to the world monitor before beginning execution (step 7). The world monitor simulates this plan and stores the expected world state after each task in the plan. Next, the executor starts the execution of the plan, checking with the world monitor after execution of each task (step 7). At the end of execution of each task, the world monitor compares the current state with the expected state of the world. If there is a difference, it triggers a flag for re-planning, so that the controller passes the control back to the planning module with the current state replacing the initial state in the planning problem (step 4). The process reiterates. The monitor also checks after each task completion as well as periodically to see if the domain description (step 9) and/or the goal state (step 11) have changed. The process designer may update the tasks in the domain description (step 13) or redefine the goal state (step 13).

During the execution of the plan, if there are any group tasks, the SHOP2 planner invokes the collaboration process module (step 5), which is a Lisp program that interfaces with the collaboration process definition repository. Currently, we assume that collaboration process definitions for any group tasks in the domain description are

stored in this repository a priori. As described in Chapter 6, the collaboration process design tool develops the collaboration process definitions in an XML format. These are transformed into a Lisp-based process description and stored in the collaboration process definition repository. The collaboration process module uses the appropriate process definition for group task execution.

The controller incorporates the parallelization by adapting the PlanParallelize algorithm (Figure 5.2) to interleaved planning and execution. In case of incomplete information, a complete plan may not be generated. Hence, the following strategy is implemented. As mentioned above, the optimal plan (probably partly generated), which is passed to controller. The controller then identifies the tasks that do not interfere with each other in the current state, and thus may be executed concurrently. A new thread of control is initiated to execute each of the concurrent tasks with synchronization between them. Upon failure of any of the tasks, re-planning is initiated from the current state.

As shown in the SPDC framework, allocation of appropriate organizational roles/agents to tasks need be performed after planning and before execution. Role resolution would be researched as an extension to the current work. It is then planned to be implemented in the prototype immediately after the step involving identification of concurrent tasks, mentioned in the above paragraph. Scheduling constraints on resources would determine whether concurrent threads of control are to be generated. If the resources are limited, tasks may be executed sequentially without violating the organizational constraints.

5.11 Evaluation

5.11.1 Evaluation: Case Studies

The feasibility of the proposed approach and the prototype system has been evaluated with case studies in the following domains: (1) home loan processing, (2) new drug

discovery, and (3) new product development. Additionally, its application to the automatic web service composition problem has been studied, by replicating the case study illustrated in [241].

Loan Processing Example : Consider the loan processing example discussed in Chapter 4. Process model generation for a particular loan application (workflow instance) is shown using the prototype system discussed above. The domain description of this example is shown in Appendix B, which consists of methods, operators, axioms, and external functions. The initial task network contains the high-level task `(process-loan-application)`, which the required business goal. A workflow instance with the initial state is shown in Figure 5.4.

```
((customer-id 101)
 (loan-application-id 565)
 (requested-loan-amount 130000)
 (property-id 969)
 (comparables-data '((property 968 100000) (property 970 125000)))
 (bank-portfolio-loans '((portfolio# 222)(portfolio# 333))))
```

FIGURE 5.4. Initial state of a loan processing application case

The final state of the workflow after plan generation is shown in Figure 5.5

```
((customer-id 101)
 (loan-application-id 565)
 (requested-loan-amount 130000)
 (property-id 969)
 (comparables-data '((property 968 100000) (property 970 125000)))
 (bank-portfolio-loans '((portfolio# 222) (portfolio# 333)))
 (credit-rating 687.5 101)
 (property-value 112500 969)
 (risk-level 4 565 101 969)
 (risk-status t nil nil 565)
 (loan-status nil 565 969 101))
```

FIGURE 5.5. Final state of a loan processing application case

Similarly, intermediate states can also be tracked which can be used for (a) process monitoring, (b) exception handling, and (c) debugging. A sample intermediate state after the `(calculate-credit-rating)` task is shown in Figure 5.6.

```
((customer-id 101)
 (loan-application-id 565)
 (requested-loan-amount 130000)
 (property-id 969)
 (comparables-data '((property 968 100000) (property 970 125000)))
 (bank-portfolio-loans '((portfolio# 222) (portfolio# 333)))
 (credit-rating 687.5 101))
```

FIGURE 5.6. Intermediate state of a loan processing application case

A given method can decompose into subtasks in several different ways depending on the preconditions. Examples of this for the `(check-risk-status ?lid ?pid ?cid)` method is shown below in Figure 5.7. The arrows indicate ordering constraints, where specified in the method description. Different alternative total-order plans would be generated depending on which preconditions are satisfied in the given state for the workflow instance at hand.

The loan processing application involves decision making tasks, such as risk evaluation. Such information is unlikely to be available a priori. Hence with interleaved planning and execution, information gathered during execution is used for further planning. In this case, the risk level is found unacceptable, and the loan application is rejected as indicated by the final state (Figure 5.5). This is illustrated below.

With the information available in the initial state, the planner can initially plan only for the tasks before `(check-risk-status ?lid ?pid ?cid)`. This results in eight possible partial orderings of tasks thus far, shown in Figure 5.8. The first plan in Figure 5.8 is shown in Figure 5.9, to illustrate the interleaving of tasks of different methods (e.g., `(!t3-appraise-property-value)` is interleaved with the subtasks of the compound task `(calculate-credit-rating)`). The ordering constraint is shown with an arrow between `(!t10-check-if-bad-risk)` and `(!t11-reject-loan)`. The preconditions and the input parameters are not shown for brevity.

Since all the plans have the same cost, one of them is chosen by the controller. Then, during the precondition evaluation of `(check-risk-status ?lid ?pid ?cid)` method, which provides additional information during execution, the plan-

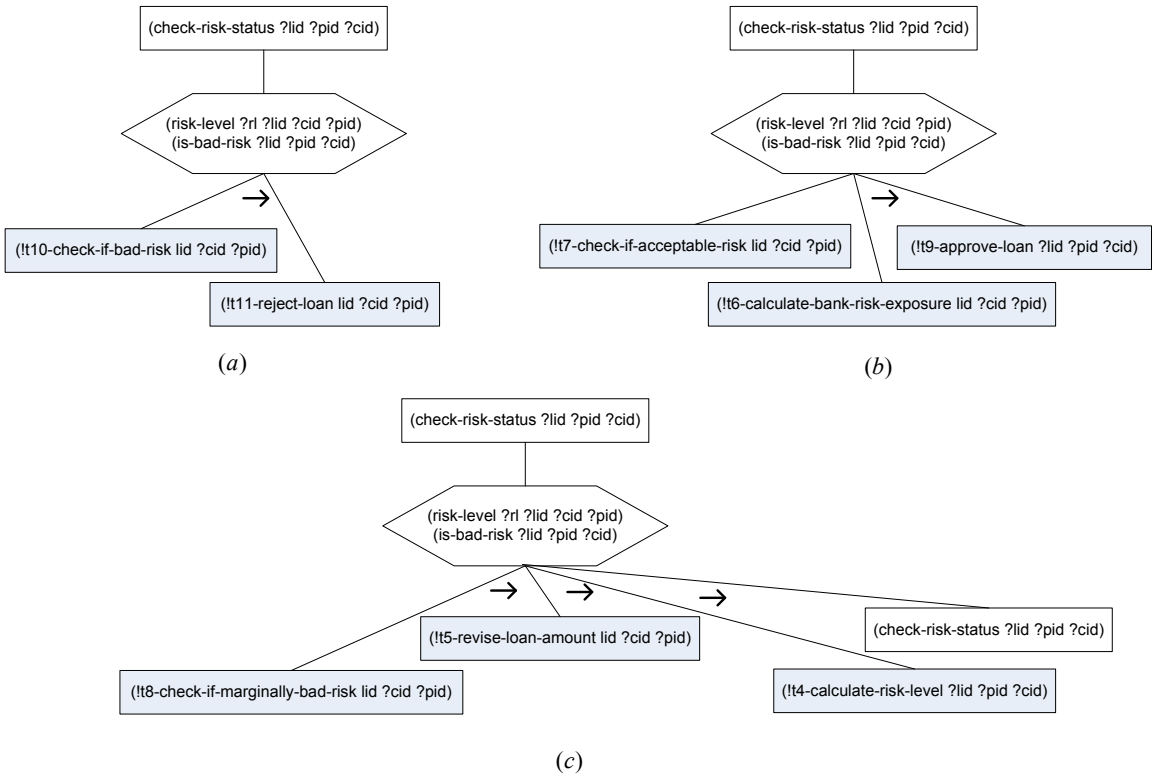


FIGURE 5.7. Multiple decompositions for `(check-risk-status ?lid ?pid ?cid)`

```
1. (!t1-calculate-credit-rating-b) -> (!t3-appraise-property-value)
    -> (!t2-calculate-credit-rating-l) -> (!!calculate-credit-rating) -> (!t4-calculate-risk-level)

2. (!t1-calculate-credit-rating-b) -> (!t2-calculate-credit-rating-l)
    -> (!t3-appraise-property-value) -> (!!calculate-credit-rating) -> (!t4-calculate-risk-level)

3. (!t1-calculate-credit-rating-b) -> (!t2-calculate-credit-rating-l)
    -> (!!calculate-credit-rating) -> (!t3-appraise-property-value) -> (!t4-calculate-risk-level)

4. (!t3-appraise-property-value) -> (!t1-calculate-credit-rating-b)
    -> (!t2-calculate-credit-rating-l) -> (!!calculate-credit-rating) -> (!t4-calculate-risk-level)

5. (!t3-appraise-property-value) -> (!t2-calculate-credit-rating-l)
    -> (!t1-calculate-credit-rating-b) -> (!!calculate-credit-rating) -> (!t4-calculate-risk-level)

6. (!t2-calculate-credit-rating-l) -> (!t3-appraise-property-value)
    -> (!t1-calculate-credit-rating-b) -> (!!calculate-credit-rating) -> (!t4-calculate-risk-level)

7. (!t2-calculate-credit-rating-l) -> (!t1-calculate-credit-rating-b)
    -> (!t3-appraise-property-value) -> (!!calculate-credit-rating) -> (!t4-calculate-risk-level)

8. (!t2-calculate-credit-rating-l) -> (!t1-calculate-credit-rating-b)
    -> (!!calculate-credit-rating) -> (!t3-appraise-property-value) -> (!t4-calculate-risk-level)
```

FIGURE 5.8. Eight possible partial orderings for a loan processing application case

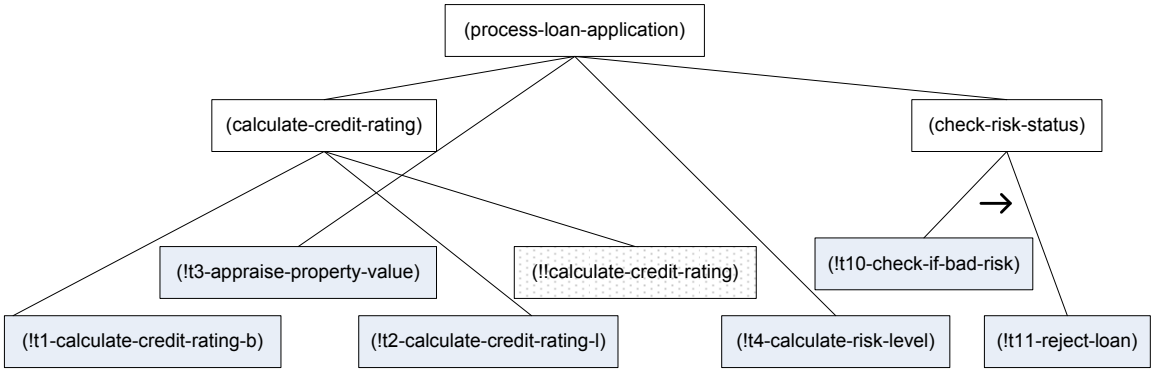


FIGURE 5.9. A plan for accomplishing `(process-loan-application)`

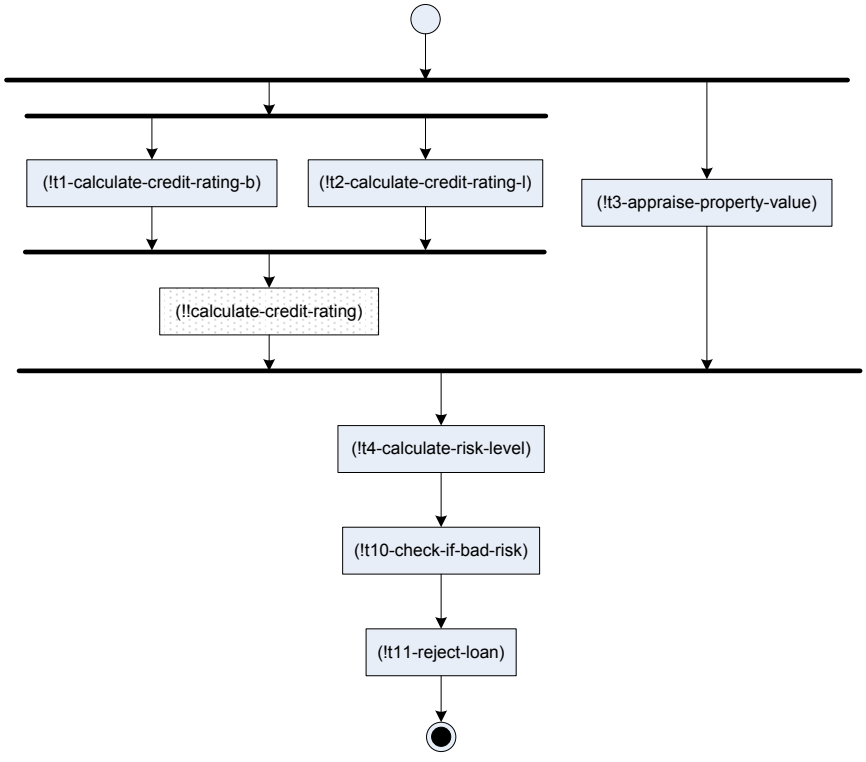


FIGURE 5.10. A concurrent plan for `(process-loan-application)`

ner chooses option (a) shown in Figure 5.7, since the risk level is found unacceptable. The tasks `(!t10-check-if-bad-risk ?lid ?pid ?cid)`, and `(!t11-reject-loan ?lid ?pid ?cid)` are then planned and executed in order. Concurrency is introduced as explained in Section 5.10. The resultant concurrent plan execution is shown in Figure 5.10.

We now show some examples illustrating the adaptability of the approach.

```
;; Task T3-e (for electronic appraisal of property value)
(:operator (!t3-e-appraise-property-value ?pid ?comp-data)
;; preconditions
((property-data ?pid)
(comparables-data ?comp-data)
(assign ?prop-val (call appraise-property-value ?pid ?comp-data)))
;; delete list
()
;; add list
((property-value ?prop-val ?pid))
;; operator cost
1)

;; Method for tasks T3 & T3-e (urgent case)
(:method (appraise-property-value ?cid ?pid ?comp-data)
;; preconditions
((applicant-data ?cid)
(property-data ?pid)
(comparables-data ?comp-data)
(is-case-urgent ?cid))
;; task list
((:ordered
(!t3-e-appraise-property-value ?pid ?comp-data))))

;; Method for tasks T3 & T3-e (regular case)
(:method (appraise-property-value ?cid ?pid ?comp-data)
;; preconditions
((applicant-data ?cid)
(property-data ?pid)
(comparables-data ?comp-data))
;; task list
((:ordered
(!t3-appraise-property-value ?pid ?comp-data))))

(:- (is-case-urgent ?cid)
urgent-case ((eval (is-case-urgent '?cid))))
```

FIGURE 5.11. Example of including a new task to the domain description

First, let us assume that the task `(!!calculate-credit-rating ?cid)`, which combines the credit ratings of the branch manager and the loan officer fails to execute. In that case, the method `(calculate-credit-rating ?cid)` uses the fallback task

`(!!manually-calculate-credit-rating ?cid)` to generate a feasible process model. This task is given a higher cost value, thus keeping it passive during normal conditions. Such an approach can account for unforseen situations (exception handling). Alternative approaches to knowledge-based exception handling have been recently studied and can be found in [194, 204, 121, 95, 173].

Next, assume that the customer immediately cancels the application for some reason, but the company policy requires to record the customer credit rating for better risk assessments in future loan applications from the same customer. This changes the goal from `(process-loan-application)` to `(calculate-credit-rating)`. The world monitor detects these changes and the controller invokes the planning module for re-planning.

Also, possibly a new task for electronic property appraisal may also be made available for customer with urgent requests. This new task can be added to the domain description along with method descriptions as shown in Figure 5.11. The dynamics of the process changes with this. If the case is tagged as urgent, both the options are now available to choose from, while in regular case, only the current method of appraisal can be allowed. Again, the world monitor helps in detecting such changes and have the controller trigger re-planning.

An external event may occur, such as the property value may change drastically during the evaluation process, causing the world state to change. This information can change the computation of the risk level. Again, the monitor informs about the change to the controller, which triggers re-planning from the current state with the updated information.

Web Service Composition Example : Generation of different multiple process alternatives in now illustrated for the case of web service composition for an online shopping example, based on the example in [241]. The results discussed in [241] have been replicated with our implementation. While HTN planning is proposed in this disser-

tation for workflow modeling in general considering automated and manual processes, this example considers a specific example of workflow design in case of automated web service composition.

TABLE 5.1. Service operators for web service S_1

Operator	Type	Input View	Output View
s_{11}	state update (transactional)	(customer id password)	(logstatus)
s_{12}	state update (transactional)	(customer id)	(logstatus)
s_{13}	state update (transactional)	(orderlist quantity)	(orderid)
s_{14}	state update (transactional)	(orderid payment)	(payment status)
s_{15}	information gathering	(item featurelist quantity)	(availability status quantity)
s_{16}	information gathering	(item featurelist)	(price)

```
(defvar *WS1-Catalog*
 '(((item DVD-player) ((brand SONY) (weight 100)) (quant 3)  (price-per-unit 250))
  ((item CD-player) ((brand SAMSUNG) (age 10)) (quant 4) (price-per-unit 150))
  ((item AnalogCamera) ((brand CANON) (age 10))  (quant 10) (price-per-unit 70))
  ((item Binoculars) ((brand ZEISS) (age 10)) (quant 10) (price-per-unit 200))
  ((item Camcorder) ((brand HITACHI) (age 10)) (quant 10) (price-per-unit 300))
  ((item DigitalCamera) ((brand SONY) (age 10)) (quant 10) (price-per-unit 450))
  ((item MP3Player) ((brand SAMSUNG) (age 10)) (quant 10) (price-per-unit 295))
  ((item TV) ((brand MAGNAVOX) (age 10)) (quant 10) (price-per-unit 700))))

(defvar *WS2-Catalog*
 '(((item TextBook) ((author Eddington) (title "New Physics")) (quant 3) (price-per-unit 45))
  ((item Magazine) ((name GQ) (date 10/02)) (quant 4) (price-per-unit 5))
  ((item MusicCD) ((artist Prince) (title "hullo")) (quant 10) (price-per-unit 20))
  ((item VideoDVD)((language English) (title "Blue Sky"))  (quant 10) (price-per-unit 10))
  ((item CasetteTape) ((title "ABBA Returns")) (quant 10) (price-per-unit 5))
  ((item BoardGame) ((title "Monopoly")) (quant 10) (price-per-unit 15))
  ((item Toy) ((name Barbie) (Manufacturer Hasbro )) (quant 10) (price-per-unit 65))
  ((item Roses) ((type Flower)) (quant 25) (price-per-unit 35))
  ((item Lily) ((type Flower)) (quant 10) (price-per-unit 15))
  ((item FictionBook) ((author Robbins) (title Carpetbaggers)) (quant 10) (price-per-unit 17))
  ((item FictionBook) ((author Keillor) (title "PHC")) (quant 10) (price-per-unit 35))))

(defvar *WS4-Catalog*
 '(((item TextBook) ((author Eddington) (title "New Physics"))  (quant 3) (price-per-unit 25))
  ((item MusicCD) ((artist Prince) (title "hullo"))  (quant 4) (price-per-unit 14))
  ((item TV) ((brand MAGNAVOX) (age 10)) ((brand MAGNAVOX) (age 10))  (quant 10) (price-per-unit 225))
  ((item CD-player) ((brand SONY) (age 10)) (quant 10) (price-per-unit 175))
  ((item VideoDVD)((language English) (title "Blue Sky"))  (quant 10) (price-per-unit 165))
  ((item Roses) ((type Flower)) (quant 5) (price-per-unit 10))
  ((item Lily) ((type Flower)) (quant 25) (price-per-unit 5))))
```

FIGURE 5.12. Catalog at each web service

A typical online shopping scenario is illustrated where one needs to shop for a set of items (possibly with interrelationship constraints) given a budget. Consider three web services S_1, S_2 and S_4, each selling a set of different items i with each item i

having a set of features j. These online services provide the following capabilities: (a) getting information regarding items and their features from product catalogs, (b) getting information regarding item availability (in terms of inventory levels), (c) getting prices of an item i, (d) placing orders, and (e) making payments. Additionally a credit card verification web service, S_3, which provides the ability to check credit cards is also modeled. Table 5.1 lists the service operators in terms of their input and output bindings for service S_1. Similar operators are defined for services S_2 and S_4. Service S_3 has an operator to check for card status.

Figure 5.12 shows the set of items and their features, prices, and current availability at each service (Note service S_i is denoted as `WSi` during encoding). For purposes of illustration, it is assumed that these are available as part of the description of each web service maintained by the service platform, which are obtained by information gathering operators s_{15} and s_{16} during execution.

A shopping service request, $SR = (Itemlist\ b)$, (a workflow instance in the shopping domain), to the platform consists of a list of items ($Itemlist = (i_1\ i_2\ \ldots)$), the feature values and quantity for each item i, and a budget b, for the purchase (example shown in Figure 5.13).

```
((shoplist (((item DVD-player) ((brand SONY) (weight 100)) (quant 2))
            ((item MusicCD) ((artist Prince) (title "hullo")) (quant 12))
            ((item VideoDVD) ((language English) (title "Blue Sky")) (quant 12))))
 (budget 3000))
```

FIGURE 5.13. Example of online shopping for three items

The above service request defines an instance of an online shopping problem and implicitly defines the state space shown in Figure 5.14, based on the web services in Figure 5.12.

The state space is shown with eight numbered states (including the initial and final states). Each state shows the key goal terms, denoted `GOAL`, with the quantity of items required in the final bundle. `CURR` denotes the items procured thus far, in

the current state.

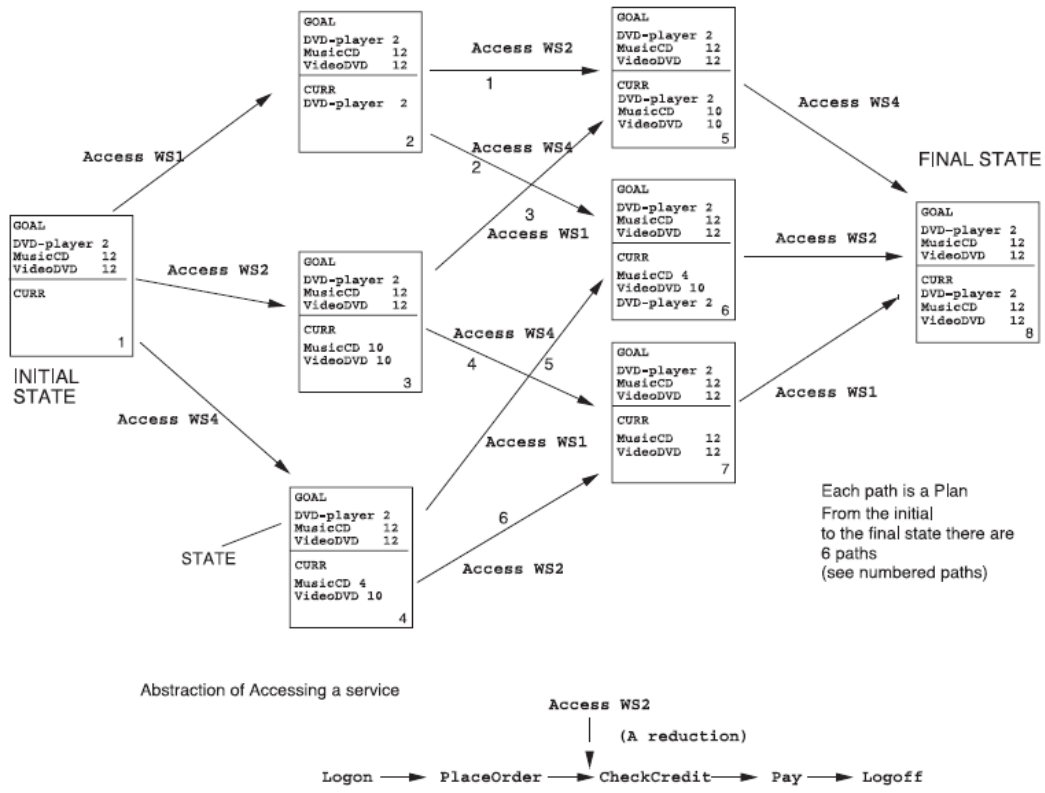


FIGURE 5.14. State space for example shopping request [241]

Thus in the initial state no items have been procured yet. State 2 shows that on accessing `WS1`, 2 units of DVD-player have been bought. Note that, the domain logic of the operator procures all the units of an item currently available in the inventory of a web service. The current availability of an item is a precondition (that is evaluated in a given state by calling the information gathering operator of the web service) before an appropriate method can be applied to that state. For clarity, each arc between two states labeled `Access WSi`, abstracts the five operators to log on, place an order, check a credit card (accessing another web service), pay for the order on approval and then log off. This is illustrated in Figure 5.14 by the abstraction hierarchy. Note that `Access WSi` is modeled as the `select_operator` method in this domain description. Also note, Figure 5.14 does not illustrate the budget or current costs in each state

(for clarity). These are also updated appropriately during the search process.

```
PLAN 1                                  PLAN 2
1:  !WS4_LOGON C1                       1:  !WS1_LOGON C1
2:  !WS4_PLACEORDER                     2:  !WS1_PLACEORDER
     (MUSICCD QUANT 4)                       (DVD-PLAYER QUANT 2)
     (VIDEODVD QUANT 10)                3:  !WS3_CHECKCARD
3:  !WS3_CHECKCARD                      4:  !WS1_PAY 500
4:  !WS4_PAY 1706                       5:  !WS1_LOGOFF
5:  !WS4_LOGOFF                         6:  !WS2_LOGON C1
6:  !WS2_LOGON C1                       7:  !WS2_PLACEORDER
7:  !WS2_PLACEORDER                          (MUSICCD QUANT 10)
     (VIDEODVD QUANT 2)                      (VIDEODVD QUANT 10)
     (MUSICCD QUANT 8)                  8:  !WS3_CHECKCARD
8:  !WS3_CHECKCARD                      9:  !WS2_PAY 300
9:  !WS2_PAY 180                        10: !WS2_LOGOFF
10: !WS2_LOGOFF                         11: !WS4_LOGON C1
11: !WS1_LOGON C1                       12: !WS4_PLACEORDER
12: !WS1_PLACEORDER                          (VIDEODVD QUANT 2)
     (DVD-PLAYER QUANT 2)                    (MUSICCD QUANT 2)
13: !WS3_CHECKCARD                      13: !WS3_CHECKCARD
14: !WS1_PAY 500                        14: !WS4_PAY 358
15: !WS1_LOGOFF                         15: !WS4_LOGOFF
```

FIGURE 5.15. Two alternative sequential service plans

The SHOP2 algorithm explores the above described state space to find a path from the initial problem state to the final state. The SHOP2 planning algorithm starts from the initial state, generates the successor states (based on the applicable methods and operators in the current state), chooses a valid successor state nondeterministically and recursively executes the algorithm. For example, the three successor states to the initial state in Figure 5.14 are generated by applying the `select_operator` method, followed by its primitive operators (based upon its reduction).

As shown in Figure 5.14, there may be multiple paths in the state space that allow transformation of the initial state into the final state. The example state space illustrates six viable paths from the initial state to the final state. Though the same web services may be accessed, different quantities of items are ordered. For example,

an order at `WS2` between states 1 and 3 consists of 10 units of `MusicCD` and `VideoDVD`, whereas an order at `WS2` between states 4 and 7 consists of 8 units of `MusicCD` and 2 units of `VideoDVD`. The order quantities differ based on what has been ordered previously and what remains to be bought. The size of this state space is determined by the number of available web services, the number of items and their quantities, the budgets and prices of the items.

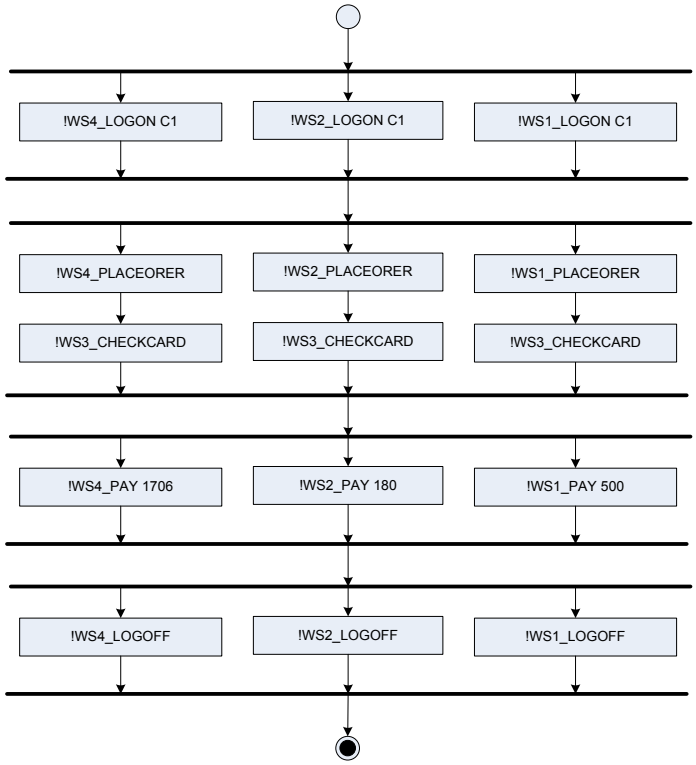


FIGURE 5.16. A concurrent plan for `PLAN 1`

Two of the six alternative sequential plans (at the service operator level), generated by SHOP2, are shown in Figure 5.15. The figure shows only the signatures of the operators (input and output bindings are not shown). The two plans differ in the items, quantity ordered and the total price to be paid. The plans differ in the access sequence of these web services. In the first plan on the left, web service S_4 is accessed first (plan Steps 1-5) followed by S_2 (Steps 6-10) and S_1 (Steps 11-15). In the plan

on the right, a different order of web services is accessed.

The two alternatives in Figure 5.15 differ in (a) which web services are accessed, (b) what order, and (c) the quantities ordered at each web service (as illustrated in the state space of Figure 5.14). For example, 12 units of `MusicCD` are ordered in the shopping request. They are procured in batches of 4 and 8 in the first alternative. In the second plan alternative, these are bought in batches of 10 and 2. The batch size bought is dictated by the available inventory at each web service at the time the order is placed. We note that different quantities of items may be ordered at different web sites based on remaining budget, availability, features and current prices leading to different plans. The choice of available web services, the items available at each service, the available quantities of each item and the prices of each item define a combinatorial state space. Different shopping strategies can be used to guide search and select feasible solutions in this space.

The sequential plans in Figure 5.15 are shown to be parallelized based on the PlanParallelize algorithm outlined in Section 5.9.2. The concurrent version of the first plan alternative is shown in Figure 5.16. Note that web services are accessed in parallel in this version with the same set of items being ordered as in sequential solution above. The numbers denote the stages of partial ordering. In the context of service planning, it is important to find a feasible plan first and then only optimize.

5.11.2 Evaluation: Workflow Pattern Based Evaluation

Workflow patterns (discussed in Section 2.9.3) were originally developed by Aalst et al. [384] as an instrument to evaluate languages supported by workflow systems. They have also been used by several authors to evaluate languages for Process-Aware Information Systems (PAIS) [103] development [102, 201, 403, 413]. The proposed generative approach to process modeling is well-suited for PAIS research, rendering the use of workflow patterns a good fit for comparing it with other languages and

formalisms developed for the same purpose.

In essence, the workflow patterns allow validation of the modeling formalisms with respect to their expressiveness. A more intuitive notion of expressiveness is sought after (as noted in [197, 384]), which takes the modeling effort into account, and is often referred to as suitability.

It is important to emphasis that the workflow patterns focus on comparing the control flow (or process) perspective as supported by different workflow modeling languages, formalisms and related products. Given that most of the formalisms assume a priori process design (process model) generated by the workflow modeler, the focus is on analyzing if the execution behavior of the pattern is supported by WFMSs incorporating those modeling languages and notations.

Hence, to evaluate the proposed approach with the right semantics, analysis is performed from two perspectives, namely, design and execution.

The design perspective deals with the capabilities of the SHOP2 AI planning formalism in supporting various control flow patterns. In order words, how can different control strategies can be incorporated in the domain description to guide the planner's search through the design space. At the design time, the preconditions and effects of activities, i.e., task dependencies determine the ordering sequence of activities. Thus, many partial ordering can be possible, as is seen with SHOP2. The threads of control are not of relevance at this stage. For example, in case of Parallel-Split pattern, the execution of an activity `A` enables the execution of activities `B` and `C`. At design time, a method with an unordered task list in SHOP2 emulates this pattern, resulting in possible process models `A -> B -> C` or `A -> C -> B`. The splitting of single thread of control into multiple threads of control is handled by the controller at execution time, who uses the minimal deordering algorithm to identify the tasks for possible concurrent execution, based on their dependency constraints. Thus, the execution perspective deals with the capabilities of the interleaved planning and execution approach where the actual execution of the process model is of concern.

Table 5.2 summarizes the results in terms of the workflow patterns from (1) the design perspective as well as (2) the execution perspective. It was checked if it is possible to realize each pattern at these two levels of abstractions. If the pattern is directly supported, it is rated +, otherwise it is rated −. For this evaluation, +/− in the execution control flow comparison, indicate that support for those patterns depend on what capabilities the controller can support for enabling concurrency. With the PlanParallelize algorithm it is possible to realize all of these patterns.

TABLE 5.2. Evaluation with workflow patterns

No.	Workflow Pattern	Build Time Semantics	Run Time Semantics
	Basic Control Flow Patterns		
1.	Sequence	+	+
2.	Parallel Split	+	+/−
3.	Synchronization	+	+/−
4.	Exclusive Choice	+	+
5.	Simple Merge	+	+
	Advanced Branching and Synchronization Patterns		
6.	Multi-choice	+	+/−
7.	Synchronizing Merge	+	+/−
8.	Multi-merge	+	+/−
9.	Discriminator	+	+/−
	Structural Patterns		
10.	Arbitrary Cycles	+	+
11.	Implicit Termination	+	+/−
	Patterns Involving Multiple Instances		
12.	Multiple Instances Without Synchronization	+	+
13.	Multiple Instances With a Priori Design Time Knowledge	+	+
14.	Multiple Instances With a Priori Runtime Knowledge	+	+
15.	Multiple Instances Without a Priori Runtime Knowledge	+	+
	State-based Patterns		
16.	Deferred Choice	+	+
17.	Interleaved Parallel Routing	+	+
18.	Milestone	+	+
	Cancellation Patterns		
19.	Cancel Activity	−	+/−
20.	Cancel Case	−	+/−

5.11.3 Summary

The focus of this research involved addressing the research question:

How can organizational processes be designed with a computational representation that will allow them to be flexible and adaptable in dynamic environments?

In this regard, the AI planning formalism was applied for business process modeling with the implementation of a prototype tool incorporating interleaved planning and execution. The prototype and the implemented case studies demonstrate the feasibility of the approach.

Flexibility and adaption of the proposed approach in dynamic environments with evolving organizational processes is illustrated in Section 5.11.1 through specific examples.

The proposed approach has also been analyzed with the workflow patterns for demonstrating the suitability and expressiveness of the modeling approach. It is seen that all the major workflow patterns are supported at the design level and also at the execution level, through the use of interleaved approach.

5.12 Conclusion

The chapter discussed the proposed approach for the design of organizational processes. The initial part of the chapter studies the role of logic-based formalisms, specifically situation calculus for process modeling. Noting its practical limitations, its variant, HTN planning is chosen as a computational formalism for generative process modeling. A design tool has been prototyped, interleaving planning and execution, that allows the process designer to generate process models that are flexible and can be adaptable in dynamic environments.

Chapter 6
Modeling of Collaboration Tasks

Fueled by organizational trends, such as increased teamwork, and technical trends, such as increased availability of networked computing infrastructure, *groupware* has emerged as promising way of supporting tasks that are carried out by groups of people in a collaborative manner. Despite this promise, however, groupware often fails to provide adequate support for day-to-day cooperative work in organizations. Studies have shown [153, 254] that inflexible support and a lack of integration between the different types of groupware as well as other enterprise applications hinder productive use of groupware.

With this in perspective, this chapter attempts to bring process structure and support to collaboration tasks, which can then be coordinated using process support tools, like GSS. Also, the proposed design bears in mind the need for integrating group activities with individual activities within a process coordination system such as WFMS.

The emerging field of collaboration engineering and their research is discusses in Section 6.1. Section 6.2 presents thinkLets, collaboration process patterns, as building blocks for collaboration process design. The collaboration process design model is presented in Section 6.3. Sections 6.1, 6.2, and 6.3 have been adopted from References [208, 87], which form the basis of the computational modeling of group tasks discussed later. The computational model for collaboration processes based on thinkLets and process algebra is given in Section 6.4. Role of case-based reasoning is discussed in Section 6.5. Finally prototype system and evaluation is discussed in Sections 6.6 and 6.7 respectively.

6.1 Collaboration Engineering for Group Task Design

With the advancement of information technology to support group work, many organizations have turned to GSS – a configurable set of software tools for structuring and focusing the efforts of teams working toward a goal. With GSS, people share, organize, and evaluate concepts, make decisions, and plan for action. GSS users may work face-to-face or across the globe. Their contributions, anonymous or identified, are available for later recall. A substantial body of research has shown that, under the right condition, teams can use GSS to enhance productivity and create organization value [180]. For example, field studies on GSS have reported labor cost reductions averaging 50% and reductions of project calendar days averaging 90% [150, 310, 88] (refer to [297, 125, 126] for a comprehensive overview of GSS research).

However, like any tool, GSSs must be wielded with intelligence and skill to produce useful outcomes. Each configuration of a GSS can be used with a variety of techniques. Thus, each combination of tool, configuration, and technique will produce different group behaviors. It can be therefore be difficult for a casual user to know how a GSS should be configured and used to create a desired outcome and realize the full potential of the GSS [58]. Many teams therefore rely of professional facilitators to manage and conduct high-value or mission-critical tasks [70, 288, 149]. Naive planning and facilitation may impair well-intended GSS efforts [84, 83], while purposeful design of effective group tasks and the selection or design of appropriate collaboration technologies to support these tasks can lead to valuable improvements in productivity [82]. For these reasons, the role of skilled facilitators has become even more pronounced in technology-supported collaboration tasks, involving not only effectively *conducting* a group session, but also the upfront *planning and design* of collaboration processes (group tasks[1]) [52, 58, 82]. Unfortunately, professional facil-

[1]Each group task in an organizational process model is considered as a collaboration process at the next level of abstraction. In this discussion, we use the terms group task and collaborative task synonymously, while noting that a collaboration process defines the dynamics of the collaborative

itators tend to be expensive. They either have to be trained in-house, or hired as external consultants. Therefore, facilitation support is not available to many groups who could derive substantial assistance for realizing these benefits.

Recently, researchers have begun to focus on findings ways for teams to wield a GSS successfully and manage their collaboration tasks for themselves with predictable results. Addressing this challenge is the domain of the emerging field of *Collaboration Engineering* [82]. Collaboration engineering is an approach for designing, modeling and deploying repeatable collaboration processes for recurring high-value collaborative tasks (group tasks) that are executed by practitioners without the ongoing intervention of facilitators. Collaboration processes designed in Collaboration engineering are processes that support a group effort toward a specific goal, mostly within a specific time frame. The process is build as a sequence of facilitation interventions that create patterns of collaboration; predictable group behavior with respect a goal. This entails a continuous reciprocal interaction without requiring co-location of participants [369].

The main thrust of the Collaboration engineering research is thus on codifying and packaging key facilitation interventions in forms that can be readily and successfully reused by groups that do not have professional facilitators at their disposal. From this standpoint, two key roles can be identified within Collaboration engineering, which help clarify the separation between the design and execution phases of a group task. A *practitioner* is a task specialist who must execute a mission critical collaborative task (e.g. requirements definition) as a part of his or her professional duties on an ongoing basis. A *group process designer* (also known as *collaboration engineer*) is a process and application domain specialist who designs collaboration processes specifically to be transferred to practitioners to run for themselves. A practitioner does not need extensive training as a facilitator, but only needs to learn the specific activities required to accomplish a particular collaboration process. A group process designer

or group task in context.

or a collaboration engineer requires both general facilitation expertise and special design and deployment skills to create processes in his or her application domain, that can be readily transferred to practitioners. This implies that a practitioner can execute the process without any further support from the group process designer, or from a professional facilitator. Table 6.1 concisely describes these roles.

TABLE 6.1. Collaboration engineering (CE) roles

Role	**Process Design**	**Process Execution**	**Expertise**
Group Process Designer / Collaboration Engineer	Repeatable, transferable process	No execution, only process transfer	Both process and application domain
Practitioner	No design	Execution based on repeatable, transferable process design	Application domain

6.2 ThinkLets: Building Blocks for Collaboration Process Design

Designs for collaboration process that can be executed by a practitioner in the field must meet special requirements, as follows:

- They must be sufficiently easy that practitioners can learn and execute them routinely, thus fitting well into the broader organizational process management framework.
- They must be sufficiently flexible to accommodate the variety of circumstances and collaborative tasks in which practitioners may use them.
- They must routinely produce results comparable to those that could be attained with the assistance of a professional facilitator.
- They must be easily synthesized and designed enabling an efficient design process for the group process designer.

- They must allow for structured design knowledge encapsulation and reuse in variety of other collaboration tasks and application domains.

To help address these requirements, collaboration engineering researchers have recently begun to codify named, packaged facilitation interventions called *thinkLets* that can produce predictable, repeatable interactions among people working toward their goal [59] (e.g., see [237, 111, 165, 336, 207]). These codified facilitation interventions are based on experiences of professional facilitators in conducting successful collaborative sessions. A few examples of thinkLets are presented in Table 6.2 (see [81] for a more elaborate description of the mechanics of these thinkLets). Each thinkLet provides a concrete group dynamics intervention, complete with instructions for implementation as part of some group process. Case studies describing such collaboration processes include an operational risk management process at an international financial services organization [387], a requirements negotiation process [50], a usability testing process [85], a mission analysis process at the U.S. Armys Advanced Research Lab [166], a knowledge elicitation process at the European Aeronautic Defense and Space company (EADS) [80], and a crisis response process in the Rotterdam Harbor in the Netherlands [23]. Additionally, thinkLets have also been used as learning modules of facilitation techniques for practitioners and novice facilitators [82] and in theoretical developments in the area of collaboration engineering [335].

Researchers have formally documented approximately 70 such distinct thinkLets [57] to date. Field experiences suggest that these 70 thinkLets account for nearly 80% of a given collaboration process design; the other 20% of group interactions need to be designed with customized thinkLets for the group task at hand. In fact, research has shown that 12 out of the 70 thinkLets often occur most frequently in collaboration process designs [207]. In this sense, thinkLets have become an expressive pattern language for group process designers, contributing reusable conceptual building blocks for designing and transferring collaboration processes [87]. In essence,

TABLE 6.2. Examples of thinkLets [81]

Name	Purpose
LeafHopper	To have a group brainstorm ideas regarding a number of topics simultaneously.
Pin-the-Tail-on-the-Donkey	To have a group identify important concepts that warrant further deliberation.
RichRelations	To have a group uncover possible categories in which a number of existing concepts can be organized.
StrawPoll	To have a group evaluate a number of concepts with respect to a single criterion.
MoodRing	To continuously track the level of consensus within the group regarding a certain issue.

a thinkLet is meant to be the smallest unit of intellectual capital required to be able to reproduce a pattern of collaboration among people working toward a goal.

6.3 Collaboration Process Design Model

The underlying rationale behind the design of collaboration processes using thinkLets is that each collaboration process (group task) can be represented as a sequence of different collaboration patterns (thinkLets) with the goal of having a process design, which when executed (possibly repeatedly), can yield a predictable behavior from the group as a whole, while creating the different constituent patterns of collaboration among the team members during the collaboration process. The ideas behind the design of collaboration process design have evolved over the past few years based on the proposed models, their implications for real world based on field trials, the shortcomings of earlier models, and the need to incorporate novel features in the model. We now present an earlier formulation of collaboration process design, mention its limitations that led to the current model of collaboration process design, and then discuss the present model, which has recently been proposed by a team of researchers (including the author).

6.3.1 Original Conceptualization of ThinkLets

Figure 6.1 shows the components of a *collaboration process*, their relationships, and the relevant details that need to be documented [208]. The name of each element is bold. Below the element's name, the documented aspects are listed. A line links related elements. Sub elements are attached below the master element. The top level element in the model represents a collaboration process, a series of actions a team takes to accomplish a goal and complete the group task. Each step in the design of a collaboration process consists of one or more *thinkLets* and *thinkLet transitions*. The model is discussed below.

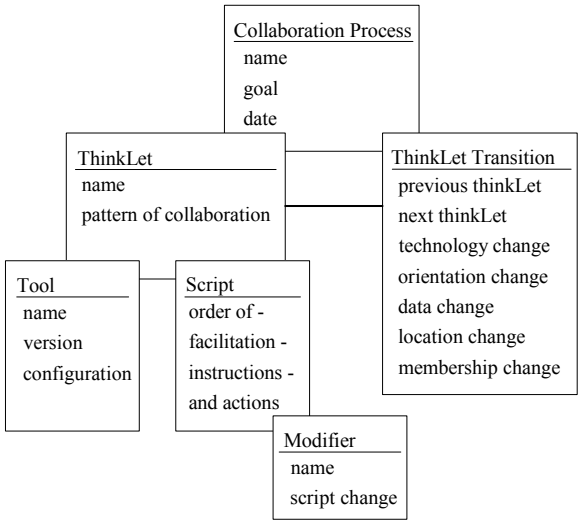


FIGURE 6.1. Definitions of the components of a thinkLet-supported collaboration process design (based on [59, 208])

As discussed in the earlier section, a thinkLet is a named, packaged, scripted collaboration activity that produces a predictable, repeatable pattern of collaboration among people working toward a goal. An initial conceptualization of thinkLets that attempted to capture this rationale is presented by Briggs et al., and proposes that a thinkLet comprises of three components: a *tool*, a *configuration*, and a *script* [59].

- The tool concerns the specific technology used to create the pattern of col-

laboration – anything from yellow stickies and pencils to highly sophisticated collaboration technologies such as GSS.

- The configuration defines how the tool is prepared (e.g., projected on a public screen), set up (e.g., configured to allow anonymous communication), and loaded with data (e.g., a set of questions to which people must respond).

- The script concerns everything a facilitator would have to do and say to a group to create the required pattern of collaboration [58, 82].

Accordingly, each differentiation in the components of a thinkLet influences the way in which people collaborate and is by definition a new thinkLet. Research indicates that small changes to, for instance, thinkLet scripts can create significant differences in group interactions (for example, see [345]).

Knowledge of these three components of a thinkLet was proposed to be sufficient for a practitioner to recreate the required pattern of collaboration [59, 58, 336, 81, 82]. Field trials with more than 200 novice trained practitioners bore out the proposition that non-facilitators who knew the tool, configuration, and script for a thinkLet could, in fact, predictably and repeatable engender the pattern of collaboration a given thinkLet was meant to produce [82]. In Table 3, the thinkLets of Table 2 are described in the tool configuration script conceptualization.

Considering thinkLets as building blocks of the collaboration process, a concept of thinkLet transitions was introduced to connect two thinkLets. The transition defines all the changes, events and actions that must take place to move people from the end of one thinkLet to the beginning of the next. A thinkLet transition design must account for some of the following aspects of changes:

- *Changes of Technology*: When one thinkLet finishes, it may be necessary to reconfigure a technology or to move to a completely different technology before the next thinkLet can begin.

- *Changes of data*: It may be necessary to transform the output of one thinkLet in some way so that it can serve as the input to the next thinkLet.
- *Changes of orientation*: It is necessary to alert team members that one activity has finished and a new one is about to start. In this alert, the team should reflect its progress in reaching their goal.
- *Changes of location*: It may be necessary for people to move from one place to another between thinkLets.
- *Changes of membership*: Sometimes it is necessary to change the composition of the team before the next thinkLet begins.

Although the importance of transitions seems obvious, it is hard to relate them to practice. Since collaboration engineering and thinkLets are often used in combination with GSS, part of the transition is automated. The role of transitions in the design of reusable, transferable and predictable collaboration process is currently a topic of further research.

Sometimes a specific combination of several thinkLets are reused frequently in different application domains. Such a sequence of thinkLets and transitions can be amalgamated into a named, reusable *compound thinkLet* [207]. It can be wielded as a single building block during process design.

Further, researchers noted that certain repeatable variations could be applied to a set of thinkLets to create a predictable change in the dynamics those thinkLets produce. These variations are termed **modifiers**. For example, all ideation/creativity thinkLets allow people to contribute any idea that comes to mind. A OneUp modifier would change the group rules for any ideation thinklet such that people many only contribute new ideas that are arguably better than the existing ideas along some dimension. This modifier may be applied to any brainstorming technique.

6.3.2 Limitations of the Original Conceptualization of ThinkLets

Although facilitators, collaboration engineers and practitioners found the initial conceptualization of thinkLets, transitions and modifiers to be useful, field experience revealed a number of drawbacks [208, 210].

First, the original concept tied a thinkLet closely to a specific technology in a specific configuration. Strictly speaking, a new thinkLet would have to be documented for any change of technology, even if the change of technology did not change the patterns of collaboration among team members. Yet, collaboration engineers in the field frequently implemented the same thinkLet with a variety of different technologies. This suggested that the tool and configuration constructs in this model might only be instances of a more fundamental concept. This is also consistent with Briggs guidelines for the development of collaboration theory, which argue the importance of concepts being independent of technology [56].

Second, the original model of thinkLets also tied a thinkLet to a particular script. The purpose of the script is to prescribe exact behavior of the facilitator to support and instruct the group. Strictly speaking, this would mean that a new thinkLet would be documented to record any changes in the things a process leader did or said. Yet, both professional facilitators and practitioners in the field often deviated from the formal thinkLet script without significantly changing the pattern of collaboration [207]. Thus, the script may only instantiate certain underlying fundamental concepts.

Third, under the original conceptualization, thinkLets were difficult to classify [208]. A reliable, detailed classification scheme for design components is an important tool for design support in any engineering discipline [210]. The root of this difficulty may have been that concept addressed practical execution details of thinkLets rather than the essence of a thinkLet. The most commonly used classification scheme organizes thinkLets based on the patterns of collaboration they engender [82]. This scheme proposes five general patterns of collaboration:

- *Diverge*: Move from having fewer to having more ideas.
- *Converge*: Move from having many ideas to a focus on and understanding of a few deemed worthy of further attention.
- *Organize*: Move from less to more understanding of the relationships among ideas.
- *Evaluate*: Move from less to more understanding of the value of ideas relative to one or more criteria.
- *Build Consensus*: Move from less to more agreement among stakeholders so that they can arrive at mutually acceptable commitments.

All thinkLets engender at least one of these patterns, so this scheme is somewhat useful for deciding which thinkLet might apply to a given situation. However, a number of thinkLets invoke multiple patterns simultaneously, so this scheme is not taxonomic. Further, it does not address issues of requisite preconditions, deliverables, available communication channels, and a host of other concepts that are important considerations when choosing one thinkLet over another.

Overall,each adaptation variation of any of the three components on a known thinkLet can become a new thinkLet in its own right. This could lead to an exponential explosion in the number of thinkLets, giving rise to much redundancy and overlap among thinkLets, and to thinkLet "dialects", where group process designers in different communities use different names for the same concepts. A large variety of thinkLets will enable a high chance of fit between thinkLets and tasks, but thinkLet dialects would make it difficult to transfer group process knowledge across the boundaries of local communities of practice. If dialects of thinkLets would be used jointly, it would increase the difficulty of the choice for a thinkLet.

This brought forth several key goals for the collaboration engineering community [209]:

- To minimize the explosion of thinkLets by identifying a stable core of conceptual thinkLets.
- To assist facilitators and collaboration engineers in choosing among the existing set of thinkLets.
- To design new thinkLets with the components of other thinkLets, being mindful not to replicate those that already exist.

Toward this end, this research work makes the following contributions:

- Formulation of a new conceptualization of thinkLets along with a team of other collaboration engineering researchers.
- Extend the proposed new conceptualization from a computational point of view.
- Propose an XML based group task representation based of the proposed formulation.
- Use case-based reasoning approach to facilitate design knowledge reuse for group task design.

6.3.3 A New Conceptualization of ThinkLets

The following discussion presents our new conceptualization of the thinkLet. This is a result of joint effort of a team of researchers, which results out of the discussions, brainstorming sessions at the First Collaboration Engineering Summer Camp held at Joslyn Castle in Omaha, Nebraska, in July 2004 and is recently presented in [87]. The new conceptualization aims at enabling collaboration engineers to:

- More easily identify the optimal thinkLets for a collaboration process design.
- More easily distinguish the relevant differences among similar thinkLets.

- More easily identify areas of collaborative endeavor for which no useful thinkLets yet exist.

- Consolidate similar thinkLets into a uniform, non-redundant base set.

Figure 6.2 presents a meta-model of a thinkLet-based process design, incorporating the newer conceptualization of thinkLets. It is represented as an object-oriented UML class diagram of a thinkLet pattern. It extends an earlier model proposed by Kolfschoten et al. [209] incorporating concepts that are required to complete thinkLets as a pattern language. The added value of a UML model, is that it shows the relations between the different elements of the thinkLet. These elements and relations are discussed below.

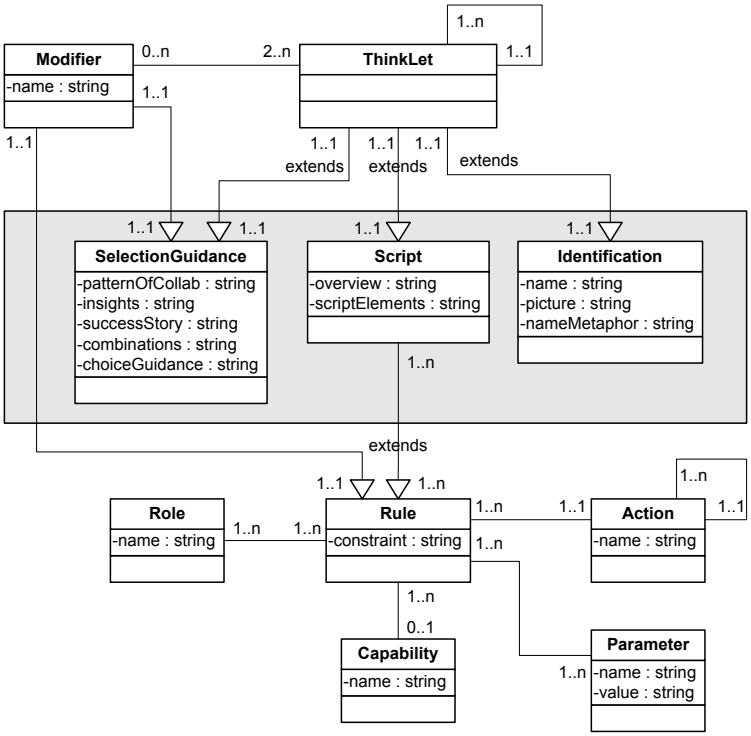


FIGURE 6.2. A class diagram of a thinkLet-supported collaboration process design (based on [87, 209])

A *thinkLet* consists of three main components, displayed in the grey section: the script, the heart of the thinkLet, selection guidance for the designer and an identification module. A thinkLet is related to itself, as shown in the right upper corner. This indicates that a thinkLet can be composed out of smaller thinkLets. This composition can have a sequential character, when several thinkLets are performed after each other or it can be the rules of several thinkLets combined into a new, compound thinkLet. The resulting compound thinkLets are more complex than the component thinkLets, but they can provide for sophisticated, subtle group dynamics that are more powerful and different than those of the component thinkLets.

A *modifier* is a rule that can be applied in some set of thinkLets to alter the interactions they produce among group members in some predictable way. For example, the OneMinuteMadness modifier can be applied to any generation thinkLet. With this modifier, the moderator stops the participants for a short time after the start of the brainstorm to discuss whether their contributions are sufficiently responsive to the brainstorming question, and to clarify the rules and constraints of the thinkLet. Afterwards, brainstorming resumes.

The *script* contains several elements. The script in the pattern is an example of how the rules of the thinkLet could be expressed to the group at execution time. A rule consists of constrained actions, which are enabled by capabilities, can be instantiated with several parameters.

The components of the UML model are the basis for the logical model of the thinkLet design patterns, which is explained below. Appendix C presents a complete pattern for the LeafHopper thinkLet. Notice that each of the sections and fields in the pattern corresponds to elements in the meta-model. Like the meta-model, the thinkLet pattern is also independent of any particular technology. A thinkLet consists of three main components: the identification, script and selection guidance. Each component will be explained below.

Identification : ThinkLets have a name, which is intended to be catchy and somewhat amusing so as to be memorable and thus easier to remember and transfer. The name also is intended to metaphorically remind a collaboration engineer of the specific group dynamics the thinkLet invokes. For example, in the LeafHopper thinkLet participants begin brainstorming several topics. The name of the thinkLet reminds its user that participants hop from topic to topic at will, making contributions here and there as inspiration strikes. To remember a thinkLet and to easily refer to it, the identification is strengthened with a picture and explanation of the metaphor. Once the metaphor is understood and remembered the name as a powerful reference, it can be used among both designers and practitioner to refer to a complex process.

Script : The script presents a bare-bones example of the instructions a practitioner or facilitator could give to the group to create the desired group interactions. The script must explain the capabilities to the team and instruct them so as to what actions should be taken and how the actions should be constrained. At design time, collaboration engineers often elaborate the bare-bones script in the thinkLet pattern into a full set of prompts and instructions for the group. The script contains an overview of the thinkLet and a set of suggested script elements.

Role : Within a thinkLet, a role is defined as a collection of rules that guide the actions of some set of participants. In some thinkLets, different participants must behave according to different rules (with different actions, constraints and/or capabilities). For example, consider two thinkLets for sorting ideas into predefined categories. In the ChauffeurSort thinkLet, one person acts as the scribe while others discuss how concepts should be organized. Thus, this thinkLet requires two roles - discussant and scribe. In the PopcornSort thinkLet, however, all participants work in parallel, each member moving ideas from a central pool into the categories where they best fit. This thinkLet has only one role - participant.

Rule : Rules describe actions that participants must execute using the *capabilities* provided to them under some set of *constraints*. In each thinkLet, individual actions are subject to constraints. For example, a brainstorming question constrains the kind of concepts a person contributes during a generation thinkLet, in that people must make contributions that are responsive to the brainstorming question.

To execute a given thinkLet, the participants must therefore become aware of the rules that are to drive their efforts. The combination of the constrained individual actions over time creates the intended dynamics within the group. For example, the rules of the FreeBrainstorm thinkLet require that each participant starts with a separate page. The FreeBrainstorming rules require that each contribution must relate to the brainstorming question, and thus participants must swap pages after each contribution. This gives rise to a breadth-variation, rather than a depth-variation of idea generation. Sometimes, technologies can be chosen to enforce one or more of the rules of a thinkLet. For example, some GSS can automatically enforce the page-swapping rule of the FreeBrainstorm thinkLet; while a group working on paper must rely on a social protocol.

Small changes to the rules that guide actions can give rise to very different interactions among participants. For example, an 'add' action guided by a 'summarize' constraint gives rise to abstraction, synthesis and generalization, while an 'add' action guided by an 'elaborate' constraint gives rise to increasingly detailed exposition of present concepts. Thus, the collaboration engineer must take care to choose constraints purposefully and to express them carefully.

Capability : The execution of a thinkLets may require tools that afford one or more capabilities. For example, the LeafHopper thinkLet mentioned above requires the following capabilities: one page for each of several brainstorming topics; participants must be able to read and contribute to all pages. It is possible to afford the same capabilities with different technologies. For example, the LeafHopper capabilities

could be implemented with flip charts, a white board or with a sophisticated GSS. The UML model therefore incorporates the concept of capabilities, while leaving the instantiation decision of how to realize those capabilities to the collaboration engineer.

Action : During the execution of a thinkLet, participants must perform certain actions as individuals - for example, add, edit, move, delete or judge ideas - using the provided capabilities. Actions play an important role in modeling the group dynamics as discussed in the later sections.

Parameter : For many thinkLets, certain information must be instantiated at process design time or execution time. For example, in a generation thinkLet, a brainstorming question must be instantiated. In an evaluation thinkLet, evaluation criteria must be defined. Therefore, parameters, which are variables with a name and value, must be instantiated before the execution of a thinkLet.

Selection Guidance : To design a collaboration process, thinkLets should be selected for a specific subtask and substep in the collaboration effort. To make this selection, the collaboration engineer has to understand the effects that the thinkLet will create. For this purpose, the thinkLet first describes the dynamics that will emerge in the group when the thinkLet is executed. To further understand, the group behavior that will occur during the execution of the thinkLet insights and success stories are included. Insights refer to tips and tricks for thinkLet implementation and execution, which support the novice facilitator or collaboration engineer when learning to use the thinkLet. Success stories provide real-life examples of the effective application of the thinkLet. Finally, for each thinkLet, proven successful combinations with other thinkLets are suggested, and choice guidance is offered in terms of 'choose this thinkLet when' and 'don't choose this thinkLet when'.

Group process designers who have a set of specific thinkLets available can therefore shift part of their attention from inventing and testing solutions to choosing known

solutions [210], thus assisting the design of group tasks.

The pattern concept described by Alexander [11, 10] has been widely adopted in the software engineering world, introduced by the gang of four [135]. For example, Lukosch and Schümmer [239] proposed a pattern language for the development of collaborative software. Patterns are successfully used in related fields such as workflow [384], communication software [323], productive software organizations [171] and for e-learning [28]. There are many parallels between the thinkLets pattern language for guided collaboration processes and Alexanders (1979) architectural design patterns. The thinkLets pattern language allows for easy communication about the collaboration process itself, and easy documentation and transfer of this process to others. It enables the rapid development of sophisticated, integrated, multi-layered collaboration processes that can improve the productivity and quality of work life for teams. As with the architectural patterns, each thinkLet pattern can create a particular dynamic within a group, but each instantiation of the pattern will differ from all other instantiations.

6.4 Computational Modeling of Collaboration Processes

In the previous sections, the approach to designing of collaboration processes was based on identifying and documenting patterns of collaboration in a group task involving concerted collaboration, so that the designer can instantiate those patterns over and over again to get repeatable and predictable results. The behavioral patterns were identified based on successful GSS field implementations, which supports the case for choosing these patterns in the first place. However, the instantiation of these patterns, as mentioned so far, really involves configuring the work environment in a particular way, so as to allow the execution of the actions involved, and thus the overall group task, in an expected manner. In case of GSS, it involves configuring the GSS toolbench, to support the actions such as `add idea`, `edit idea`, and so forth as

well as allowing explicit synchronization between these actions in a manner that will allow the particular collaboration pattern to surface. Hence, from a process designer's point of view, it would be valuable to explicitly encapsulate the group interaction in terms of a minimal set of actions. On one hand, when the process designer needs to merely choose and the appropriate patterns and instantiate them, this information about the group dynamics can be abstracted away and be hidden from the process designer. On the other hand, when the process designer needs to customize a pattern to the group task at hand or want to even design new patterns of collaboration, the information on group interaction can allow him or her to create or edit the interaction, simulate it, store it, so as to allow easy execution with the supporting groupware tools.

With the aim to capture the dynamics of group interaction in a collaboration pattern, a computational model of such an interaction is proposed that is amenable for easy simulation and extensible for execution. Any kind of model is a simplified representation of the real world and, as such, includes only those aspects of the real-world system relevant to the problem at hand. In this case, the actions from the thinkLet patterns are chosen to be included in the model to represent the dynamics of group interaction. Given the inherent concurrent nature of collaboration processes, we draw on the concurrent programming literature to model the group tasks as Labeled Transition Systems (LTS) as described in the following section. This model is then captured in a generic process definition (also having an XML representation) for each pattern. A case-based reasoning system is then used to support reusing these elements of design to create an executable definition for each group task at hand.

6.4.1 Modeling Group Tasks as Labeled Transition Systems

The basic structure of concurrent programs is now described, using the terminology in the concurrent programming community. As noted in the concurrent programming

literature, the execution of a sequential program (or subprogram) is termed as a *process* and the execution of a concurrent program thus consists of multiple processes. The state of a process at any point in time consists of the values of explicit variables, declared by the programmer, and implicit variables such as the program counter and contents of data/address registers. As a process executes, it transforms its state by executing statements consisting of one or more *atomic actions* that make indivisible state changes. Examples of atomic actions are uninterruptible machine instructions that load and store registers. A more abstract model of a process is simply to consider a process as having a state modified by indivisible or atomic actions. Each action causes a transition from the current state to the next state. The order in which actions are allowed to occur is determined by a transition graph that is an abstract representation of the program. In other words, processes can be modeled as *state transition systems*.

Given this basic structure of concurrent programs, an analogy can be drawn between group tasks and concurrent programs, at a higher level of abstraction. Each participant in the group task can be said to follow a sequence of actions during the execution of the group task, similar to a sequential program. Each of these actions would then result in indivisible state changes. For example, a new idea suggested by a participant would add to the group knowledge about the topic at hand and change the state of the group task. Thus, when all the participants are jointly engaged in the group activity, the concurrent execution of different actions by different participants would display a particular group behavior that is analogous to the behavior displayed by the execution of a concurrent program consisting of multiple processes.

Correspondence Between Finite State Processes and Labeled Transition Systems : Essentially, a state transition system, from the theoretical computer science literature, is an abstract machine used in the study of computation, consisting of a set of states and transitions between states. In a state transition system the set of transitions is

not necessarily finite, or even countable. In a state transition system, transitions do not form a function, but a relation between the states, and therefore, there may be zero or more than one transition out of a given state, with the same input. State transition systems with a finite number of states and transitions can be represented as directed graphs. There are at least two basic types of state transition systems: "labeled" or "unlabeled". A *Labeled Transition System* (LTS) representation is used for modeling group tasks, wherein the transitions represent the actions taken by the participants to change the state of collaboration. LTSs can be represented diagrammatically and can be displayed using tools such as the *Labeled Transition System Analyzer* (LTSA) [243]. The graphical form of LTS allows for easy visualization of the individual participant actions as well as the overall group interaction. LTS is supported by a formal, but simple process algebraic notation proposed by Magee et al. [243], called *Finite State Processes* (FSP) to textually specify LTSs. The LTSA tool, developed by researchers at the Imperial College London [243], allows for inputing FSP notation to produce and analyze the corresponding LTSs. FSP is thus a specification language with well-defined semantics in terms of LTSs, which provides a concise way of describing LTSs. Each FSP expression E can be mapped onto a finite LTS.

The association of state machines with process algebra, as used in the formalization of FSP, was proposed by Milner [264], who describes an operational semantics for a Calculus of Communicating Systems (CCS) using LTS. While the semantics of FSP are derived from the CCS approach, the syntax is built on Hoare's [181] Communicating Sequential Processes (CSP).

A LTS is represented with the following diagrammatic conventions. If processes in a composition have actions in common, these actions are said to be *shared*. Shared actions are the way that process interaction is modeled in LTS. While unshared actions may be arbitrarily interleaved, a shared action, must be executed at the same time by all the processes that participate in that shared action. The initial

state is always numbered 0, and transitions are labeled with action names and are always drawn in a clockwise direction. The FSP process algebraic notation is used to describe process models.Every FSP description has a corresponding state machine (LTS) description. We will show an example of modeling a group task using the LTS and FSP representations later in this section. Now, a brief overview of the notation of FSP is given below. The full language definition of FSP may be found in [243].

FSP introduces several operators, including an action prefix, choice, recursion, an end state, and parallel composition. Some of the noteworthy FSP operators are summarized in the bulleted list below along with their brief semantics.

If `x` and `y` range over actions, and `P` and `Q` range over FSP processes, FSP introduces the following important operators:

- *Action prefix* "`->`": `(x->P)` describes a process that initially engages in the action `x` and then behaves exactly as described by the auxiliary process `P`.
- *Choice* "`|`": `(x->P|y->Q)` describes a process which initially engages in either `x` or `y`, and whose subsequent behavior is described by auxiliary processes `P` or `Q`, respectively.
- *Recursion*: the behavior of a process may be defined in terms of itself, in order to express repetition.
- *End state* "`END`": describes a process that has terminated successfully and cannot perform any more actions.
- *Sequential composition* "`;`": `(P;Q)` where `P` is a process with an `END` state, describes a process that behaves as `P` and when it reaches the `END` state of `P` starts behaving as the auxiliary process `Q`.
- *Parallel composition* "`||`": `(P||Q)` describes the parallel composition of processes `P` and `Q`.

- *Relabeling* "/": `/{newlabel`$_1$`/oldlabel`$_1$`,...,newlabel`$_n$`/oldlabel`$_n$`}` is the general form of relabeling, which can be applied to a process to change the names of action labels.

- *Hiding* "\": the hiding operator `\{a`$_1$`,...,a`$_x$`}` removes the action names `a`$_1$`,...,a`$_x$ from the alphabet of `P` and makes these concealed actions "silent". These silent actions are labeled `tau`. Silent actions in different processes are not shared.

- *Interface* "@": the interface operator `@{a`$_1$`,...,a`$_x$`}` hides all actions in the alphabet of `P` not labeled in the set `a`$_1$`,...,a`$_x$.

The important concepts in FSP modeling are summarized here. In the FSP notation, a process is defined by one or more local processes separated by commas. The definition is terminated by a full stop. A process can optionally be parameterized and have relabeling, hiding, and alphabet extension parts.

A composite process is distinguished from a primitive process by prefixing its definition with `||`. Composite processes are constructed using parallel composition, relabeling, priority, and hiding. Process labeling and sharing are specialized forms of relabeling. The replication and conditional constructs are purely syntactic conveniences.

The sequences of actions produced by the execution of a process (or a set of processes) is known as a *trace*. In general, processes may have many possible execution traces. Such traces are possible because of the choice operator.

In order to model processes and actions that can take multiple values, both local processes and action labels may be indexed in FSP, thus greatly increasing the expressive power of the notation. The finite range of values of the indices ensures that the models described in FSP are finite and thus potentially mechanically analyzable.

The *alphabet* of a process is the set of actions in which it can engage. A process may only engage in the actions in its alphabet; however, it may actions in its alphabet

in which it never engages. In FSP, the alphabet of a process is determined implicitly by the set of actions referenced in its definition. The alphabet extension construct allows for inclusion of additional actions in the alphabet of the process that are not referenced in its definition.

Semantics of Concurrency : Concurrency is next discussed in the context of concurrent programs and collaborative tasks, noting their implications.

The execution of a concurrent program consists of multiple processes active at the same time, where each process is the execution of a sequential program. A process progresses by performing a sequence of instructions using a processor. If there are multiple processors available, instructions from a number of processes, equal to the number of physical processors, can be executed at the same time, representing what is referred to as *real* or *parallel* concurrent execution.

However, if the number of processors are less than the active processes, the available processors are switched between processes (as shown in Figure 6.3), referred to as *pseudo-concurrent* execution or an *interleaving* of the instruction sequences from each individual process. The processor switching does not affect the order of instructions executed by each process.

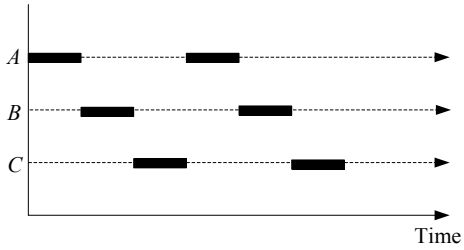


FIGURE 6.3. Interleaved execution of three concurrent processes on a single processor

The terms *parallel* and *concurrent* are used interchangeably while modeling concurrent programs using LTS and there is no distinguishing between real and pseudo-concurrent execution since, in general, the same programming principles and techniques are applicable to both physically (real) concurrent and interleaved execution.

In fact, concurrent execution is always modeled as interleaved, whether or not implementations run on multiple processors.

This concurrent execution model in which processes perform actions in an arbitrary order at arbitrary relative speeds is referred to as an *asynchronous* model of execution. It contrasts with the *synchronous* model in which processes perform actions in simultaneous execution steps, sometimes referred to as lock-step.

The execution of group tasks can be viewed analogously. The actions performed by each participant member of the group are similar to a sequence of instructions of a process. A group consisting of multiple participants (like multiple processes) is then similar to a concurrent program. A participant is responsible for his/her own actions, like a physical processor executing a process. Hence, in this sense, the execution of a group task is an example of *real* concurrent execution.

Drawing from the discussion above, an asynchronous model of execution is used to modeling group tasks, without any loss of generality. The actions, which a participant can execute independently within the context of the group task, are interleaved with actions of other participants, whereas actions which depend on the *real* concurrent execution are *shared*.

Two issues are particularly noteworthy in this context:

- *Speed of execution*: In case of design of concurrent programs, the relative speed at which one process executes relative to another is not modeled, but is assumed that processes execute at arbitrary relative speeds (meaning that a process can take an arbitrarily long time to proceed from one action to the next). This results in allowing the programs to works correctly, independent of the number of processors and the scheduling strategy, thus abstracting away the execution time. This has the disadvantage that one can say nothing about the real time properties of programs but has the advantage that other properties can be verified independently of the particular hardware configuration, which

is important for the portability of concurrent programs.

In case of design of group tasks, abstracting away execution time is important for the flexibility in designing collaboration processes for different meeting environments, such as shown in Figure 6.4. Groupware supported groups can work in a number of modes. They can work in a dedicated meeting room; they can work as individuals linked by electronic media in a virtual meeting; they can be located in several meeting rooms that are connected electronically. They can all work at the same time (synchronously), or different members can work at different times (asynchronously). They can work in small groups of four or five, or they can work in very large groups. This was also briefly mentioned in Chapter 2 in Section 2.4.1. Collaborative tasks in each of these working modes can be designed with the concurrency model in LTSs, since is presents no constraints related to time and place.

As noted in Section 6.3, thinkLets have been used as collaborative process patterns, primarily for group tasks occurring in the concerted mode (i.e., same time/synchronous mode). In this dissertation,this type of group interaction is considered. But the proposed model based on concurrent processes is extensible and applicable to cases which involve asynchronous collaborative tasks.

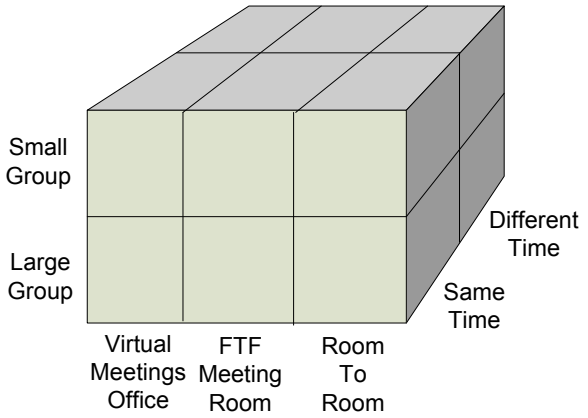


FIGURE 6.4. Types of meeting environments (based on [298])

- *Interleaved Concurrency Model*: In case of concurrent programs, one of the issues is whether it is necessary to model the situation in which actions from different processes can be executed simultaneously by different processors in addition to the situation in which concurrency is simulated by interleaved execution? In case of interleaved concurrency model, an action a is concurrent with another action b if a model permits the actions to occur in either the order $a \rightarrow b$ or the order $b \rightarrow a$. Since time is not represented in the model, the fact that the event a actually occurs at the same time as event b does not affect the properties we can assert about concurrent executions.

  The same logic applies to group tasks. Each participant is free to perform the individual actions in the overall group task as long as they are obey the sequential constraints and are synchronized at various points in time, where the concerted effort is needed for the group to make progress. For example, in a voting session, after each member has voted and cast the ballot, a discussion involving all group members can throw light on the results. Such a discussion action would need to be synchronized as against the voting action, which can take place independently for each participant.

Computational Representation of ThinkLets : In the previous sections, it was argued that thinkLets form the building blocks in designing collaboration processes. A computational model in terms of LTS and FSP is proposed for thinkLets that can potentially be executed using different software components enabling each of the actions involved therein. The aim of representing thinkLets in this manner is to capture the group dynamics in successful behavioral patterns, demonstrated in field experiments, to guide the design of similar collaboration tasks in different domains, in a preditable and repeatable manner. ThinkLets have been documented in a computationally informal manner (e.g., LeafHopper thinkLet example in Appendix C).

In order for a thinkLet to be executable, the most important component, namely

the actions, need to captured in a machine friendly way. This has driven our modeling of thinkLets as LTS using the FSP notation discussed above. Figure 6.6 below shows the LeafHopper thinkLet in terms of LTS representation. The corresponding FSP representation is shown in Figure 6.5. For simplicity, we have shown the example with a group of two participants, working on two topics. Obviously, this model can be extended to any number of participants and topics.

```
const GROUPSIZE = 2 // change the value depending on the group size
range MEMBERS = 1..GROUPSIZE

const TOPICS = 2 // change the value depending on the number of topics
range TOPIC_LIM = 1..TOPICS

// Role 1: Participant
PARTICIPANT = (choose_topic[u:TOPIC_LIM] -> {topic[u].read_idea, topic[u].add_idea} -> PARTICIPANT |
end_ideation -> END).

// Role 2: Practitioner
PRACTITIONER = (decide_end_ideation -> END).

// Group Task (composition of member processes)
minimal ||LEAFHOPPER = (forall[i:MEMBERS] p[i]:PARTICIPANT || q:PRACTITIONER)
/{end_ideation/{p[x:MEMBERS].end_ideation, q.decide_end_ideation}}.

menu RUN = {p[j:MEMBERS].choose_topic[k:TOPIC_LIM],
p[a:MEMBERS].topic[b:TOPIC_LIM].read_idea, p[c:MEMBERS].topic[d:TOPIC_LIM].add_idea, end_ideation}
```

FIGURE 6.5. FSP representation of the LeafHopper thinkLet

TABLE 6.3. Static representation of the LeafHopper thinkLet

No.	Local Action	Global Action	Performer
1.	p[1].choose_topic[1]	p[1].choose_topic[1]	PARTICIPANT 1
2.	p[1].choose_topic[2]	p[1].choose_topic[2]	PARTICIPANT 1
3.	p[2].choose_topic[1]	p[2].choose_topic[1]	PARTICIPANT 1
4.	p[2].choose_topic[2]	p[2].choose_topic[2]	PARTICIPANT 2
5.	p[1].topic[1].read_idea	p[1].topic[1].read_idea	PARTICIPANT 1
6.	p[1].topic[2].read_idea	p[1].topic[2].read_idea	PARTICIPANT 1
7.	p[2].topic[1].read_idea	p[2].topic[1].read_idea	PARTICIPANT 2
8.	p[2].topic[2].read_idea	p[2].topic[2].read_idea	PARTICIPANT 2
9.	p[1].topic[1].add_idea	p[1].topic[1].add_idea	PARTICIPANT 1
10.	p[1].topic[2].add_idea	p[1].topic[2].add_idea	PARTICIPANT 1
11.	p[2].topic[1].add_idea	p[2].topic[1].add_idea	PARTICIPANT 2
12.	p[2].topic[2].add_idea	p[2].topic[2].add_idea	PARTICIPANT 2
13.	p[1].end_ideation	end_ideation	PARTICIPANT 1
14.	p[2].end_ideation	end_ideation	PARTICIPANT 2
15.	q.decide_end_ideation	end_ideation ($\star$)	PRACTITIONER

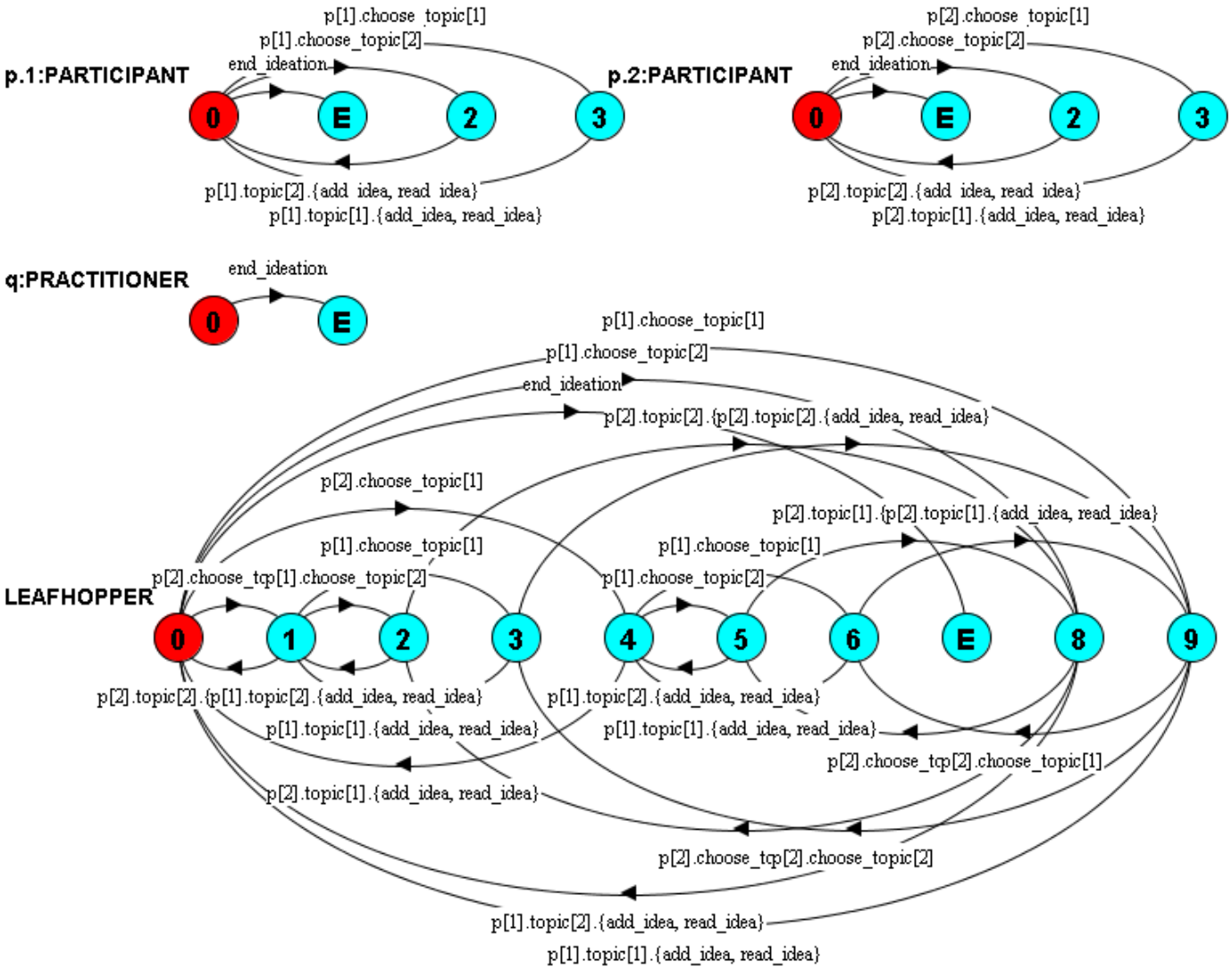


FIGURE 6.6. LTS representation of the LeafHopper thinkLet

While modeling thinkLets as labeled transition systems, as noted above, each participant engages in an action, which is referred to as a *local action*. Each local action corresponds to a global action. In cases where the individual actions are independently performed, the global actions have the same labels as the local actions. Two or more individual actions can be synchronized for either or both of two reasons: (1) to expend a concerted effort on certain part of the group task, (2) to synchronize the overall group activity at certain stage of the task. In such cases, the local actions are said to be "shared", and their global counterparts are shown with a separate label representing the shared action. The local and global actions for the LeafHopper thinkLet are recorded in a separate table, Table 6.3, for ease of explanation. The LTS representation in Figure 6.6 is shown with global labels for all actions.

From Table 6.3, it can be seen that actions such as `p[x].topic[x].read_idea` and `p[x].topic[x].add_idea`, are independently carried out by the participants (indicated by same local and global action labels). The `p[x].end_ideation` action for the participants is a shared action along with the `q.decide_end_ideation` action of the practitioner. The shared global action `end_ideation`, which occurs concurrently. The star marked next to the practitioner's global action indicates that the practitioner actively executes this action, while the participants passively follow.

The LTSA tool can be used to simulate the group dynamics for the LeafHopper thinkLet. Various traces can be observed. A sample trace of the thinkLet (in terms of the global actions) could be:

```
p[1].choose_topic[1] -> p[1].topic[1].add_idea -> p[2].choose_topic[2] ->
p[2].topic[2].add_idea -> p[1].choose_topic[2] -> p[1].topic[2].read_idea
-> end_ideation.
```

Modeling of another thinkLet, namely the Concentration thinkLet is now illustrated. In a nutshell, the goal of the Concentration thinkLet is organize and clean up the list of ideas generated by a group to eliminate redundant, ambiguous, or overlapping ideas. Figure 6.8 below shows the Concentration thinkLet in terms of LTS

```
const GROUPSIZE = 2 // change the value depending on the group size
range MEMBERS = 1..GROUPSIZE

// Role 1: Participant
PARTICIPANT = (wait -> choose_similar_ideas -> discuss -> PARTICIPANT |
end_organize -> END).

// Role 2: Practitioner
PRACTITIONER = (choose_category -> moderate_discussion ->
{edit_ideas, remove_ideas} -> PRACTITIONER | decide_end_organize -> END).

// Group Task (composition of member processes)
minimal ||CONCENTRATION = (forall[i:MEMBERS] p[i]:PARTICIPANT || q:PRACTITIONER)
/{choose_similar_ideas/{p[a:MEMBERS].choose_similar_ideas},
practitioner_choose_category/{p[a:MEMBERS].wait, q.choose_category},
discussion/{p[a:MEMBERS].discuss, q.moderate_discussion},
end_organize/{p[n:MEMBERS].end_organize, q.decide_end_organize}}.

menu RUN = {practitioner_choose_category, choose_similar_ideas, discussion,
q.edit_ideas, q.remove_ideas, end_organize}
```

FIGURE 6.7. FSP representation of the Concentration thinkLet

representation. The corresponding FSP representation is shown in Figure 6.7. For simplicity, we have shown the example with a group of two participants, working on two topics, which is extensible to any number of participants. The local and global actions for the Concentration thinkLet are recorded in a separate table, Table 6.4, for ease of explanation. From this table, the mappings between local actions and global actions mentioned earlier can be noted for all cases, as explained below.

TABLE 6.4. Static representation of the Concentration thinkLet

No.	Local Action	Global Action	Performer
1.	p[1].wait	practitioner_choose_category	PARTICIPANT 1
2.	p[2].wait	practitioner_choose_category	PARTICIPANT 2
3.	q.choose_category	practitioner_choose_category ($\star$)	PRACTITIONER
4.	p[1].choose_similar_ideas	choose_similar_ideas ($\star$)	PARTICIPANT 1
5.	p[2].choose_similar_ideas	choose_similar_ideas ($\star$)	PARTICIPANT 2
6.	p[1].discuss	discussion ($\star$)	PARTICIPANT 1
7.	p[2].discuss	discussion ($\star$)	PARTICIPANT 2
8.	q.moderate_discussion	discussion ($\star$)	PRACTITIONER
9.	q.edit_ideas	q.edit_ideas	PRACTITIONER
10.	q.remove_ideas	q.remove_ideas	PRACTITIONER
11.	p[1].end_organize	end_organize	PARTICIPANT 1
12.	p[2].end_organize	end_organize	PARTICIPANT 2
13.	q.end_organize	end_organize ($\star$)	PRACTITIONER

Examples of independent local actions include `q.edit_ideas` and `q.remove_ideas`,

carried out by the practitioner (indicated by same local and global action labels).

Local actions `p[1].discuss`, `p[2].discuss`, and `q.discuss` are shown as shared (with global label `discussion`), involving active participation of all group members (indicated by global actions marked with star). They represent the concerted action of discussing similar ideas.

The local actions `p[1].choose_similar_ideas`, and `p[2].choose_similar_ideas` are also shared (with global label `choose_similar_ideas`), but actively engaging only the participant role members in choosing similar ideas (indicated by global actions marked with star). The practitioner is idle during this action (implied implicitly). This activity is shared for synchronization in the group task, when one of the participants is proposing similar ideas in the chosen topic.

In a similar case, local actions `p[1].wait`, `p[2].wait`, and `q.choose_category` are also shared (with global label `practitioner_choose_category`), but actively engaging only the practitioner (indicated by global action marked with star). The participants are waiting for the category to be chosen (indicated explicitly). This action is shared for synchronization in the group task.

Collaboration Process Design : Based on the new conceptual model of thinkLets (see Figure 6.2), as well as the FSP and LTS-based computational formalism, a collaboration process (group task) can then be modeled as a sequence of these patterns to accomplish the group goal.

For example, consider the drug discover example. In the library concept development phase shown in Figure 4.6, the evaluation of proposals is a group activity, which can be modeled as the following sequence of thinkLets:

```
Plus-Minus-Interesting -> Concentration -> MultiCriteria -> Crowbar
```

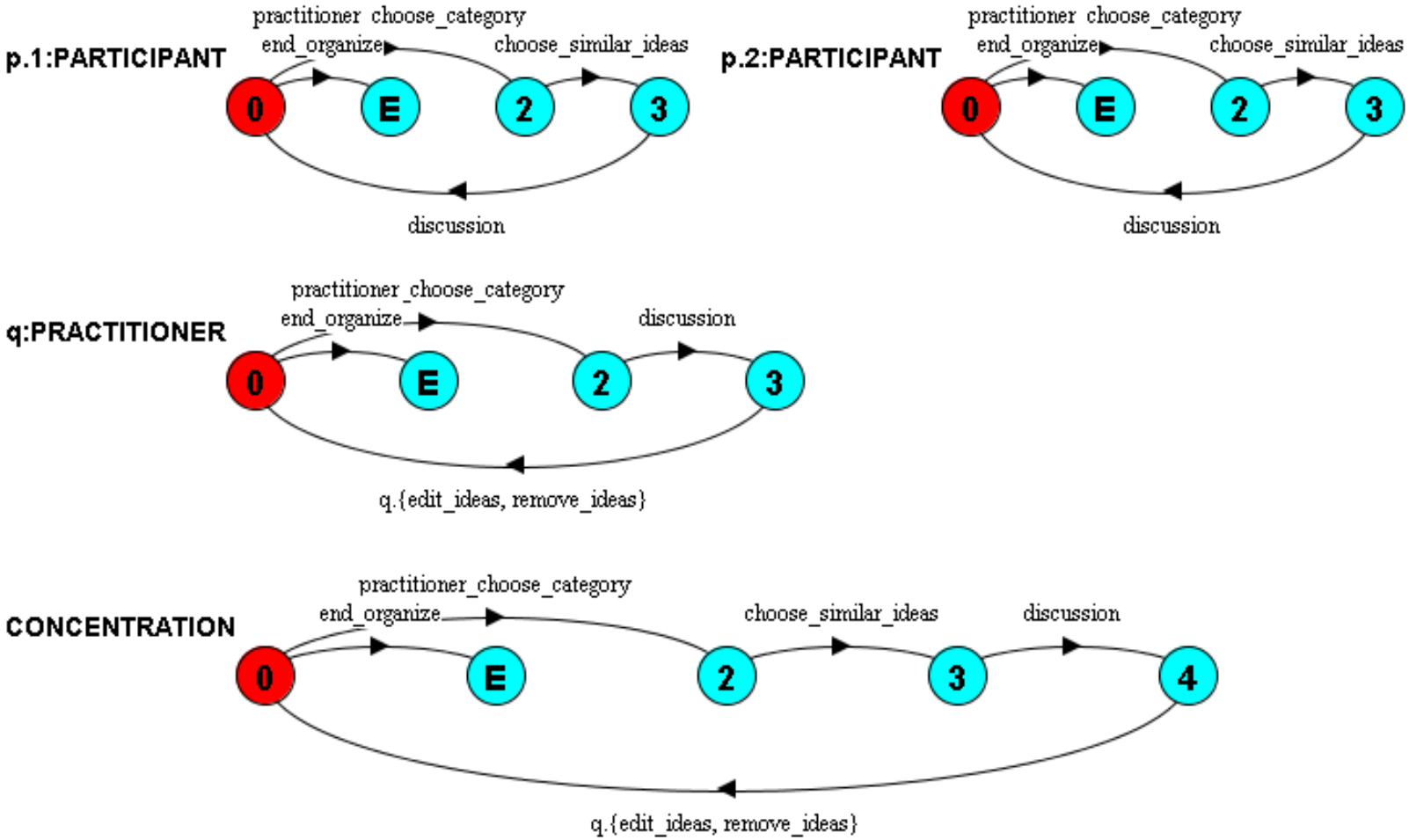


FIGURE 6.8. LTS Representation of the Concentration ThinkLet

The goal of this group task is to evaluate a set of proposals based on certain criteria. The group involves the chemistry division head and the current project leaders. It is explained below how the sequence of thinkLets can help in accomplishing the goal.

Plus-Minus-Interesting thinkLet allows the group to evaluate each proposal, based on their strengths and weaknesses. Next, Concentration thinkLet as mentioned before allows the group to produce a clean list of pros and cons for each proposal, eliminating redundancy. Multicriteria thinkLet then allows the group to evaluate (rate) each of the proposals based on the set of predefined criteria. Finally, crowbar helps to analyze the results of the evaluation to facilitate selection of the proposals. The details of these thinkLets can be found in [57].

The overall design of a collaboration process can be perceived as a staged process, seeking solutions to each of the questions below, in order:

1. What is the goal (deliverables) of the collaboration task?
2. What steps are involved in accomplishing the goal?
3. What kind of group behavior can be expected at each step towards achieving the goal?
4. Which process pattern(s) would be best suited in producing such behavior?
5. How would the group interactions be structured within each of the patterns to achieve expected behavior?

The first three questions are best supported by the expertise of the group process designer (collaboration engineer). Next, it is necessary to determine the sequence of process patterns as well as the structure of the group dynamics involved therein. Currently, group process designers have to perform these steps in an ad hoc manner. Even once the thinkLet patterns are determined, they cannot be directly materialized

using groupware tools such as GSS, due to lack of a computational model underlying these patterns. The proposed computational model can allow capturing the intended group dynamics during the execution of these patterns. Moreover, in order to allow process designers to build on their previous experiences of designing successful group meetings, it is needed to provide a design tool that can leverage this process knowledge, at the same time allowing them to customize it to their needs, and focus on the design using patterns and group interactions. A case-based reasoning approach to collaborative process design can facilitate these advantages, and will be presented in the next section, followed by a discussion of a prototype system that has been built demonstrating the utility of this approach to group process design.

6.5 Process Design Support With Case-Based Reasoning

Case-based reasoning is a problem solving technique based on the hypothesis that reasoning is reminding. That is, problem solving utilizes past experiences. CBR is a computational approach that supports explicit reuse of partial and possibly incomplete, experiential knowledge (stored as cases) in solving ill-structured and complex cognitive tasks, such as design. Past knowledge may be reused to explore the process design space and synthesize new solutions. CBR systems have proven useful in domains with weak models and a large body of unstructured, experiential knowledge [1, 224]. Successful CBR-based systems have been developed for supporting both product and process design [175, 242, 244, 271, 275].

A case-based approach to a process design activity such as group task modeling is feasible because of the recurrence of similar collaboration tasks in different organizational contexts, such as meetings involving project review, risk assessment, requirements gathering, and so forth. Hence, a case-based reasoning (CBR) approach [211], which consists of case retrieval, case reuse, case adaptation and case verification tasks, is proposed to support process model reuse during group task design. The CBR ap-

proach to group task design has been prototyped in a design tool.

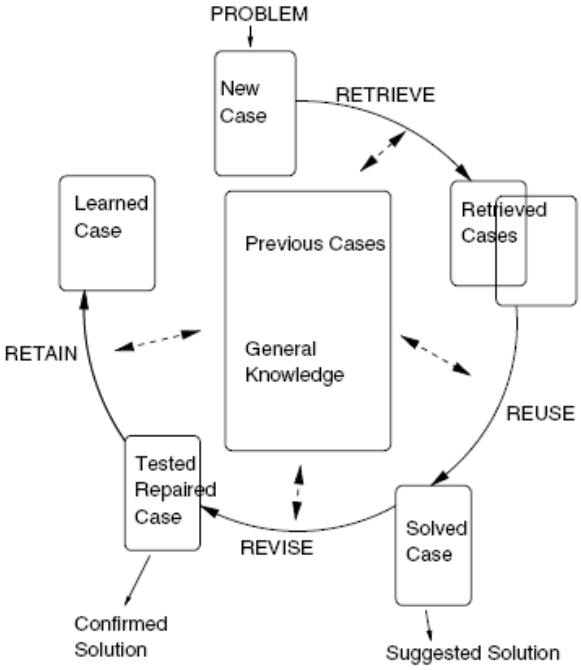


FIGURE 6.9. The case-based reasoning cycle [211]

The generic CBR problem solving cycle (called CBR cycle) is illustrated in Figure 6.9, and consists of the following steps [1]: retrieval of relevant (similar) cases from the repository based on cues derived from problem requirements, reuse of applicable cases to suggest solutions to a new problem, knowledge-based revision of relevant cases, testing-based verification and rule-based validation to ensure correctness, and retention of past solutions and failures to enable learning. A variety of application-specific case retrieval (during the reuse phase) and case adaptation (during the revise phase) techniques are discussed in [211].

The term *case* in CBR is used to refer to a unit of knowledge in the given application domain. In this context, group process definitions based on FSP and thinkLet representations are the cases. These process definitions are of two types, namely, the group process schema (called prototypical cases) and group process instances (called instance-level cases). Prototypical cases have generic information about the process

patterns that needs be customized to the application domain for instantiation purposes. Instance-level cases are essentially execution traces of prototypical cases for well-defined inputs. When a group process definition is to be modified or updated, it is stored as a new prototypical case in the repository. Each case is chosen to be represented using the Extensible Markup Language (XML) for interoperability with existing and new collaboration process support systems.

Cases in collaboration process modeling contain both structured and unstructured information. The FSP representation of thinkLets, representing action interactions contain domain independent, well-structured information in order to enable execution. Information based on the conceptualization of the process patterns (thinkLets) such as brainstorming prompt is mostly domain dependent and unstructured, and allows actual instantiation of the patterns. Thus, both structured and unstructured information is important to collaboration task design.

6.5.1 Indexing Group Process Cases

The collection of group process cases (both prototypical and instance level) provides a potentially large space to be searched when a process designer is looking for a relevant case to quickly design a new group process. In order to find a relevant group process schema or an instance in an appropriate amount of time, the cases need to be discriminated properly. In general, cases need to be assigned labels that can be designate design situations in which cases are likely to be useful. These labels are referred to as *indices* to individual cases [244].

The selection of which parts of a case representation, or which of its features should act as its indices is a problem of defining the indexing vocabulary [211]. The indexing vocabulary can be taken directly from the contents of cases (i.e., attributes used to represent cases).

In case of group process cases, a relational model of indices based on the meta-

model of conceptualization of thinkLets, presented in Section 6.3.3 (refer Figure 6.2), is used. The attributes described therein, namely, identifier, script, role, rule, capability, action, parameter, and selection guidance form the indexing vocabulary for the case library. This indexing vocabulary is used for case retrieval in the prototype system described in the next section.

6.6 Prototype System

The prototype system developed assists the process designer in creating process definitions for group activities, which can be used in the overall process design. As indicated in Section 5.10, the collaboration process module in the integrated planning and coordination prototype takes a Lisp-based process definition as an input to execute a group activity. Such collaboration process definitions are obtained by translating the XML-based process definitions created during group task design.

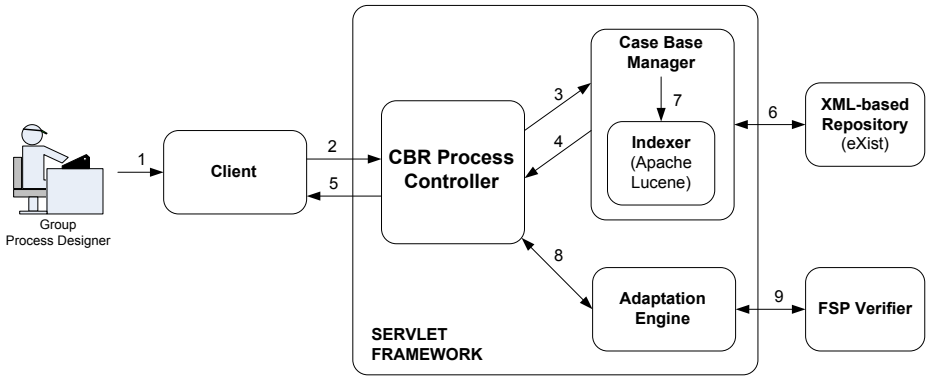


FIGURE 6.10. Prototype system architecture

The prototype architecture is shown in Figure 6.10. It is based on the CODAW system architecture for workflow model management [242], although it has been oriented for group task model management. Different phases of the CBR cycle are supported including retrieval, reuse, composition, adaptation and verification. Numbers in the figure indicate the different steps supported with respect to the CBR

cycle. The CBR process controller module is the central engine coordinating different processes. Case retrieval is initiated by a query (step 1), cues are generated from the problem description (step 2), and case base manager is invoked for case retrieval (step 3), retrieval is performed at the case base manager (step 6), retrieved cases are ranked (step 4) and returned to the user via the controller (step 5). Case addition maybe supported by the indexing and repository update (steps 7 and 6). Cases can also Alternatively, retrieved cases may be adapted and verified (steps 8 and 9), and the solution returned (step 5). Figure 6.10 also illustrates the different open source software technologies used in the prototype. Text retrieval is supported by the Apache Lucene engine. XPath and XQuery mechanisms of XML-based search are supported by the eXist native XML database framework. Different modules are implemented as Lisp and Java servlets. The FSP verifier and adaptation modules are currently being developed.

6.7 Evaluation

The focus of this research involved addressing the research question:

> *How can collaborative activities be designed with a computational representation that will allow them to have predictable, and repeatable results as well as facilitate integration with individual activities in organizational processes?*

The proposed representation for modeling collaboration processes is based on grounded theory in the concurrent programming literature. Also, with a goal to ensure predictable and repeatable results from this model, the model is guided by the use of behavioral process patterns, thinkLets. These patterns in turn have originated from lab and field experiments with a goal of documenting successful patterns in order to achieve predictability and repeatability [237, 111, 165, 336, 207]. In order

to validate the FSP and LTS-based model of collaboration processes, it would be valuable to replicate the outcomes and results of these studies.

Also, the feasibility of the idea of utilizing a case-based reasoning approach for assisting collaboration engineers to design new processes based on previous process knowledge is validated through building a proof-of-concept prototype system. Several collaboration process definitions in the domains of (1) new drug discovery, (2) new product development, and (3) home loan processing have been created using this system. To understand the utility of the approach, in each of these domains, collaboration tasks were designed using available GSS tool, namely GroupSystems II (earlier version called Cognito) process design studio. For comparison purposes, the appropriate thinkLets were chosen and then customized for these group tasks using the GSS setup tool. It was observed that the time required to design the group tasks using the studio was at least three times greater than that required using the design tool. This preliminary investigation has triggered confidence in this approach, which needs to validated through formal experimentation. The system also needs to be tested empirically to ensure the quality of the generated process designs.

Thus, the proposed computational model as well as the CBR-based process design approach opens several avenues for future research, which can allow for stronger validation of the ideas presented here. These potential research directions are discussed below to guide future research:

- One of the advantages of proposing a computational model for group tasks is that such a declarative approach can build upon recent model-driven architecture advancements [168, 206] to support design of new collaboration tools. The proposed process definition, which embeds the domain independent computation model to represent group dynamics as well as domain specific information to instantiate process patterns, can then be leveraged by such GSS tools to design and execute group tasks on-the-fly. This indicates research potential for

this work in the area of software engineering and development. In this regard, current developments in the area of modeling interactions on the web are being studied closely [148, 147, 235].

- Creating GSS tools which can support the use of proposed process definition will allow testing the model in lab and field studies. Previous studies designed by manually configuring GSS tools to emulate thinkLet process patterns, can now be designed efficiently with the new tools taking the group process definition as an input. The experimentation would be along two lines: (1) studying the design process by using design support tools to create process definitions (such as the prototype developed), as against manual design using conventional tools, and (2) the results and outcomes of the group session to compare and analyze the utility of the proposed approach.

- The prime motivation for creating a model for collaboration processes was supporting interleaved individual and group activities with seamless information flow between them within a BPMS. This has been demonstrated on a prototypical level by the current work. More intensive study of this aspect is necessary, e.g. studying the types of information flow supported. Such a study is envisioned with the development of new GSS tools incorporating the proposed model.

- The literature in the area of GSS, CSCW, and collaboration engineering, is vast and records several lab and field studies. Many of these studies document details of the collaboration activity involved. These studies can be looked upon in retrospection to analyze which process patterns could be applicable for these cases. This will allow to populate the library of group process definitions over a wide range of domains. Again, these cases can be validated in the field to test for efficiency of the proposed design approach.

- ThinkLets are patterns that are primarily concerned with concerted type of collaboration in a face-to-face scenario. The proposed approach is a step toward automated facilitation, since the group process design task of the facilitator is now done a priori by using CBR-based support tools; also the facilitator scripts and prompts are encapsulated as actions, executed by a practitioner not needing any facilitator skills. This has implications for supporting distributed or virtual collaboration. Developing patterns for successful distributed group sessions, and supporting them with the proposed model will be a challenging research direction to pursue.

6.8 Conclusion

The chapter discussed the proposed approach to the design of collaboration task, based on process patterns called thinkLets. A computation model for thinkLets, has been proposed, which is based on Finite State Processes and Labeled Transition Systems. A design tool has been prototyped that allows the process designer to quickly assemble a process definition based on the above model, as well as integrate it with the overall process model consisting of individual activities.

Chapter 7
Conclusion

This dissertation work has focused on process modeling and design aspect of organizational processes. With the aim to achieve better process coordination and management, this research focused on addressing the following two research questions:

1. *How can organizational processes be designed with a computational representation that will allow them to be flexible and adaptable in dynamic environments?*

2. *How can collaborative activities be designed with a computational representation that will allow them to have predictable, and repeatable results as well as facilitate integration with individual activities in organizational processes?*

7.1 Contributions

In pursuit toward addressing the above research questions, a number of significant contributions of this work to the information systems field can be noted:

- A structured top-down approach to enable design of business processes in an explicit manner, including the design of collaboration activities

- Explicit process representation for describing organizational processes, with a detailed model for collaboration processes

  - Exploratory study of logic-based approaches to business process modeling, including situation calculus

  - Proposed declarative process representation for overall workflow modeling based on HTN planning, a variant of situation calculus

  - Use of collaboration process patterns, thinkLets, for modeling of group tasks with better conceptualization
  - Formalization of collaboration processes using finite state processes and labeled transition systems, to model thinkLets

- Use of computational procedures (AI planning) to automate the generation of process design alternatives
- Interleaved planning and execution approach for flexibility and adaptability of workflow modeling
- Use of computational procedures (case-based reasoning) to facilitate design knowledge reuse for group task design
- Generative approach to design facilitating extensions to model-driven systems development and process enactment

The declarative approach to organizational process design outlined in this dissertation was motivated by the need to develop robust workflow models, facilitate knowledge reuse, and minimize the total costs of workflow management. The benefits of such a declarative approach include the following [309]:

- Design of production and custom workflows can be supported within the same framework supporting reuse of shared knowledge within a given application domain.
- A wide variety of constraints may be considered to guide the workflow design. Multiple alternative process models can be generated for a given set of tasks and constraints, from which a feasible model may be selected based on optimization metrics.

- The task of causality-based task sequencing is decoupled from resource-based task sequencing. Thus, the task of workflow design can be decoupled from runtime scheduling. Scheduling-based concurrency may then be considered at runtime. This allows for a better understanding of the workflow design rationale and focus on process (re)engineering with task dependencies.

- The approach supports process knowledge reuse. Interactive and automated plan composition based on plan fragments retrieved from a repository may be supported.

- Interleaving planning and execution supports complex overall control flow behavior, however, with simpler individual process models. This supports easy troubleshooting, failure repair and incremental extension.

- The process models generated are guaranteed to be correct with respect to the domain knowledge provided to the planning mechanism as they are based on resolution mechanisms of logic-based theoretical foundations.

In a similar vein, the collaboration process model was developed to address the need for seamless flow of information across processes, automated facilitation, and predictable and reliable group task design. It has the following implications for business process design:

- With the underpinnings of a computational formalism, collaboration processes can be integrated within the overall process management framework, allowing for better information flow across activities.

- The computational model can also formally verify and reason about the properties of a group task.

- Collaboration tasks can be designed efficiently with the help of design support tools incorporating the computational model.

- New collaboration tools can be developed through model-driven systems development.

7.2 Limitations and Implications

The major bottleneck envisioned in implementing the SPDC framework in organizations is the cost of capturing the business knowledge declaratively. Though this endeavor is challenging, recent focus on developing ontologies in the context of research on the Semantic Web, suggests that developing shared declarative process representations may be a viable trend in the near future [146]. Also, recent research on process ontologies [158, 159], ontology theories [203], process knowledge acquisition [198, 199] and semantic web services [78, 79, 9] are particularly relevant in this regard. The need for declarative requirements gathering has also been considered in the development of generic software systems [62]. Our work has been inspired by [63, 64] on knowledge-base task modeling and task reuse in a domain specific manner. It has to be noted that though the declarative knowledge gathering may seem a limitation at first instance, the potential advantages of adaptability and flexibility based on reasoning mechanisms outweigh the procedural ways of defining processes, which primarily work under the assumption of requirement completeness.

7.3 Closing Remarks

Process coordination and management are central to modern day organizations. With increasing dynamic nature of business processes, the need for designing flexible and adaptable processes as well as interleaving individual and collaborative activities through a central process coordination management system is evident. This dissertation has addressed this issue with a view of assisting the process designer by providing a top-down structured design cycle, supported with automated process

model generation techniques for overall process design as well as a CBR-based approach for collaboration process design. The suggested declarative perspective for workflow modeling has important implications for IT and business managers. For example, workflow design and other knowledge management activities in the organization, such as Six Sigma need to aligned to avoid duplication in process modeling efforts. Overall, the benefits of good process design can be realized only through effective process controlling and management. This calls for more work and research in the area of business process management to develop unique approaches, as well to harness methodologies and techniques from related disciplines to address the currently open research problems in this field.

Appendix A
Situation Calculus

A.1 The Language of Situation Calculus

The basic elements of the first-order[1] logical language $\mathcal{L}_{sitcalc}$ of situation calculus include the following, in addition to the standard alphabet of logical symbols of first-order languages (including $\neg$, $\forall$, $\exists$) [308]:

- Three disjoint sorts: *action* for actions, *situation* for situations, and a catch-all sort *object* for everything else, depending on the domain of application. There are countably infinitely many individual variable symbols of each sort, so it is possible to quantify over actions, situations, and objects. They are denoted by $\forall\vec{x}$, $\forall a$, and $\forall s$, where, $\vec{x}$, a, and s denote object sets, action and state variables respectively.

- Two function symbols of sort *situation*:

  1. A constant symbol S_0, denoting the initial situation.

  2. A binary function symbol $do : action \times situation \rightarrow situation$. The intended interpretation is that $do(a, s)$ denotes the successor situation resulting from performing action a in situation s.

- A binary predicate symbol $\sqsubset$: $situation \times situation$, defining an ordering relation on situations. The intended interpretation is that situations are action histories; thus, $s \sqsubset s'$ means that s is a proper subhistory of s'.

[1] Strictly speaking, the language of situation calculus is second order. However, the only second order logical sentence used is the induction axiom (see later discussion). Therefore, the situation calculus language is restricted to a standard sorted first order sublanguage (with equality) to express all other axioms of the domain theories.

- A binary predicate symbol $Poss : action \times situation$. The intended interpretation of $Poss\,(a, s)$ is that it is possible to perform action a in situation s.

- For each $n \geq 0$, countably infinitely many predicate symbols with arity n, and sorts $(action \cup object)^n$, used to denote situation independent relations.

- For each $n \geq 0$, countably infinitely many function symbols of sort $(action \cup object)^n \rightarrow object$, used to denote situation independent functions. For each $n \geq 0$, a finite or countably infinite number of function symbols of sort $(action \cup object)^n \rightarrow action$, called *action functions*, used to denote actions. Notice that function symbols taking values of sort *object* are distinguished from those – the action functions – taking values of sort *action*. As discussed later, the latter are distinguished by the requirement that they be axiomatized in a particular way by what are called *action precondition axioms*.

- For each $n \geq 0$, a finite or countably infinite number of predicate symbols with arity $n+1$, and sorts $(action \cup object)^n \times situation$. These predicate symbols are called *relational fluents* and are used to denote situation dependent relations.

- For each $n \geq 0$, a finite or countably infinite number of function symbols of sort $(action \cup object)^n \times situation \rightarrow action \cup object$. These function symbols are called *functional fluents* and are used to denote situation dependent functions.

It can be seen that in the language $\mathcal{L}_{sitcalc}$, only two function symbols, S_0 and *do*, are permitted to take values in sort *situation*. Also, it can be noted that the *object* sort may denote various objects in the domain of application and can be further classified into sub-sorts, if necessary. In our current framework for workflow modeling, the objects include data entities of the domain. It is possible to extend this framework to include the role entities in the domain description and use description logic (DL) [29], a subset of first-order logic to reason about domain relationships between these entities.

A.2 Axiomatizing in the Situation Calculus

The dynamics of a chosen domain is specified by defining the causal laws in language $\mathcal{L}_{sitcalc}$ along with foundational axioms of the language. A *basic action theory* in $\mathcal{L}_{sitcalc}$ is any collection of axioms $\mathcal{D}$ of the following form[2] [308]:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$$

where,

- Σ are the foundational axioms for situations.
- $\mathcal{D}_{ap}$ is a set of action precondition axioms.
- $\mathcal{D}_{ss}$ is a set of successor state axioms for functional and relational fluents.
- $\mathcal{D}_{una}$ is the set of unique names axioms for actions.
- $\mathcal{D}_{S_0}$ is a set of first-order sentences that are uniform[3] in S_0, so that S_0 is the only term of sort *situation* mentioned by the sentences of $\mathcal{D}_{S_0}$. $\mathcal{D}_{S_0}$ functions as the initial theory of the world (i.e., the one we start off with, before any actions have been "executed") for solving the workflow problem instance at hand.

Σ, $\mathcal{D}_{ap}$, $\mathcal{D}_{ss}$, and $\mathcal{D}_{una}$ are explained below.

Foundational Axioms for Situations(Σ): The foundational axioms are domain independent and provide the basic properties of situations in any domain specific axiomatization of particular fluents and actions [321]. Σ includes the following four axioms:

[2] A *basic action theory* must also satisfy an additional *functional fluent consistency property* that provides a sufficient condition for preventing a source of inconsistency in a functional fluent f's successor state axiom. (Refer [308] for details.)

[3] A formula of $\mathcal{L}_{sitcalc}$ is uniform in σ iff it does not mention the predicates $Poss$ or $\sqsubset$, it does not quantify over variables of sort *situation*, it does not mention equality on situations, and whenever it mentions a term of sort *situation* in the situation argument position of a fluent, then that term is σ.

- $do\left(a_1, s_1\right) = do\left(a_2, s_2\right) \supset a_1 = a_2 \wedge s_1 = s_2$

  This is a unique names axiom for situations; two situations are the same iff they result from the same sequence of actions applied to the initial situation, i.e., if they denote identical histories.

- $\left(\forall P\right).P\left(S_0\right) \wedge \left(\forall a, s\right)\left[P\left(s\right) \supset P\left(do\left(a, s\right)\right)\right] \supset \left(\forall s\right)P\left(s\right)$

  This axiom is second-order induction on situations, and has the effect of limiting the sort *situation* to the smallest set containing S_0, and closed under the application of the function *do* to an action and a situation.

- $\neg s \sqsubset S_0$

- $s \sqsubset do\left(a, s^{'}\right) \equiv s \sqsubseteq s^{'}$

  Both of the above axioms together are designed to capture the concept of a subhistory of action sequences. Here $s \sqsubseteq s^{'}$ is an abbreviation for $s \sqsubset s^{'} \vee s = s^{'}$. The relation $\sqsubset$ provides an ordering relation on situations. Intuitively, $s \sqsubset s^{'}$ means that the action sequence $s^{'}$ can be obtained from the sequence s by adding one or more actions to the front of s.

Action Precondition Axiom ($\mathcal{D}_{ap}$): An action precondition axiom of $\mathcal{L}_{sitcalc}$ is a sentence of the form:

$$Poss\left(A\left(\vec{x}\right), s\right) \equiv \Pi_A\left(\vec{x}, s\right)$$

where A is an n-ary action function symbol, and $\Pi_A\left(\vec{x}, s\right)$ is a formula that is uniform in s and whose free variables are among $\vec{x}, s$. The uniformity requirement on Π_A ensures that the preconditions for the executability of the action A are determined by the current situation s, which addresses the qualification problem for actions.

Successor State Axiom ($\mathcal{D}_{ss}$) (SSA): Solving the frame problem [343] with respect to the effect axioms of actions for the given domain results in the SSAs for that

domain [320]. SSA characterizes the immediate effects of an action in terms of the fluents, whose truth values we know are changed by the execution of the action in a given situation. Effects of an action on a fluent in a situation can either be positive (it becomes true or remains true) or negative (becomes false or remains false).
A SSA for an $(n+1)$-ary relational fluent F is a sentence of $\mathcal{L}_{sitcalc}$ of the form:

$$F\left(\vec{x}, do\left(a, s\right)\right) \equiv \Phi_F\left(\vec{x}, a, s\right)$$

where $\Phi_F\left(\vec{x}, a, s\right)$ is a formula uniform in s, all of whose free variables are among $\vec{x}, a, s$. As for the action precondition axioms, the uniformity of Φ_F guarantees that the truth value of $F\left(\vec{x}, do\left(a, s\right)\right)$ in the successor situation $do\left(a, s\right)$ is determined entirely by the current situation s, and not by any other situation. In systems and control theory, this is often called the *Markov property*.
A SSA for an $(n+1)$-ary functional fluent f is a sentence of $\mathcal{L}_{sitcalc}$ of the form:

$$f\left(\vec{x}, do\left(a, s\right)\right) = y \equiv \phi_f\left(\vec{x}, y, a, s\right)$$

where $\phi_f\left(\vec{x}, y, a, s\right)$ is a formula uniform in s, all of whose free variables are among $\vec{x}, y, a, s$. As for relational fluents, the uniformity of ϕ_f in the successor state axioms for functional fluents guarantees the Markov property.

Unique Names Axioms for Actions ($\mathcal{D}_{una}$): For distinct action function names A and B of $\mathcal{L}_{sitcalc}$,

$$A\left(\vec{x}\right) \neq B\left(\vec{y}\right).$$

Identical actions have identical arguments:

$$A\left(x_1, \ldots, x_n\right) = A\left(y_1, \ldots, y_n\right) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n.$$

Appendix B
Loan Processing Domain

```
;; planning domain
(defdomain loan-process (

;; Top-level method
(:method (process-loan-application)
;; preconditions
()
;; task list
((:unordered
(:task calculate-credit-rating)
(:task !t3-appraise-property-value ?pid ?comp-data)
(:task !t4-calculate-risk-level ?lid ?cid ?pid)
(:task check-risk-status ?lid ?cid ?pid))
))

;; Method for tasks T1 & T2
(:method (calculate-credit-rating)
;; preconditions
()
;; task list
((:unordered
(:task !t1-calculate-credit-rating-b ?cid)
(:task !t2-calculate-credit-rating-l ?cid)
(:task calculate-credit-rating ?cid))))

;; Task T1
(:operator (!t1-calculate-credit-rating-b ?cid)
;; preconditions and inputs/knowledge preconditions
((applicant-data ?cid)
(account-data ?cid)
(assign ?cr-b (call calc-credit-rating-b ?cid)))
;; delete list
()
;; add list
((credit-rating-b ?cr-b ?cid))
;; operator cost
1)

;; Task T2
(:operator (!t2-calculate-credit-rating-l ?cid)
;; preconditions
((applicant-data ?cid)
(credit-history ?cid)
(assign ?cr-l (call calc-credit-rating-l ?cid)))
;; delete list
()
;; add list
((credit-rating-l ?cr-l ?cid))
;; operator cost
1)

;; Method for combining outputs of tasks T1 & T2
(:method (calculate-credit-rating ?cid)
;; preconditions
((credit-rating-b ?cr-b ?cid)
```

```
(credit-rating-l ?cr-l ?cid)
(calc-credit-rating ?cr-b ?cr-l ?cid))
;; task list
((:ordered
(!!calculate-credit-rating ?cid)))
;; preconditions
((credit-rating-b ?cr-b ?cid)
(credit-rating-l ?cr-l ?cid))
;; task list
((:ordered
(!!manually-calculate-credit-rating ?cid))))

;; Operator for combining outputs of tasks T1 & T2
(:operator (!!calculate-credit-rating ?cid)
;; preconditions
((credit-rating-b ?cr-b ?cid)
(credit-rating-l ?cr-l ?cid)
(assign ?cr (calc-credit-rating ?cr-b ?cr-l ?cid)))
;; delete list
((credit-rating-b ?cr-b ?cid)
(credit-rating-l ?cr-l ?cid))
;; add list
((credit-rating ?cr ?cid))
;; operator cost
0)

;; Operator for manually combining outputs of tasks T1 & T2
(:operator (!!manually-calculate-credit-rating ?cid)
;; preconditions
((credit-rating-b ?cr-b ?cid)
(credit-rating-l ?cr-l ?cid)
(assign ?cr (call manually-enter-credit-rating ?cr-b ?cr-l ?cid)))
;; delete list
((credit-rating-b ?cr-b ?cid)
(credit-rating-l ?cr-l ?cid))
;; add list
((credit-rating ?cr ?cid))
;; operator cost
2)

;; Task T3
(:operator (!t3-appraise-property-value ?pid ?comp-data)
;; preconditions
((property-data ?pid)
(comparables-data ?comp-data)
(assign ?prop-val (call appraise-property-value ?pid ?comp-data)))
;; delete list
()
;; add list
((property-value ?prop-val ?pid))
;; operator cost
1)

;; Task T4
(:operator (!t4-calculate-risk-level ?lid ?cid ?pid)
;; preconditions
((credit-rating ?cr ?cid)
(property-value ?prop-val ?pid)
(loan-amount ?loan-amt ?lid)
(assign ?rl (call calculate-risk-level ?cr ?prop-val ?loan-amt)))
;; delete list
()
```

```
;; add list
((risk-level ?rl ?lid ?cid ?pid))
;; operator cost
1)

;; Method for tasks T10 & T11
(:method (check-risk-status ?lid ?pid ?cid)
;; preconditions
((risk-level ?rl ?lid ?cid ?pid)
(is-bad-risk ?lid ?pid ?cid))
;; task list
((:ordered
(:task !t10-check-if-bad-risk ?lid ?pid ?cid)
(:task !t11-reject-loan ?lid ?pid ?cid))))

;; Method for tasks T6, T7, T9
(:method (check-risk-status ?lid ?pid ?cid)
;; preconditions
((risk-level ?rl ?lid ?cid ?pid)
(is-acceptable-risk ?lid ?pid ?cid))
;; task list
((:ordered
(:unordered
(:task !t7-check-if-acceptable-risk ?lid ?pid ?cid)
(:task !t6-calculate-bank-risk-exposure ?lid ?pid ?cid))
(:task !t9-approve-loan ?lid ?pid ?cid))))

;; Method for tasks T4, T5, T8
(:method (check-risk-status ?lid ?pid ?cid)
;; preconditions
((risk-level ?rl ?lid ?cid ?pid)
(is-marginally-bad-risk ?lid ?pid ?cid))
;; task list
((:ordered
(:task !t8-check-if-marginally-bad-risk ?lid ?pid ?cid)
(:task !t5-revise-loan-amount ?lid ?pid ?cid)
(:task !t4-calculate-risk-level ?lid ?cid ?pid)
(:task check-risk-status ?lid ?pid ?cid))))

;; Task T5
(:operator (!t5-revise-loan-amount ?lid ?pid ?cid)
;; preconditions
((property-value ?prop-val ?pid)
(risk-level ?rl ?lid ?cid ?pid)
(risk-status ?br ?ar ?mr ?lid)
(loan-amount ?loan-amt ?lid)
(assign ?revised-loan-amount (call revise-loan-amount ?prop-val ?rl ?mr ?loan-amt)))
;; delete list
((loan-amount ?loan-amt ?lid))
;; add list
((loan-amount ?revised-loan-amount ?lid))
;; operator cost
1)

;; Task T6
(:operator (!t6-calculate-bank-risk-exposure ?lid ?pid ?cid)
;; preconditions
((property-value ?prop-val ?pid)
(loan-amount ?current-loan-amt ?lid)
(bank-portfolio-loans ?bp)
(assign ?risk-exposure (call calculate-bank-risk-exposure ?prop-val ?current-loan-amt ?bp)))
;; delete list
```

```
()
;; add list
((bank-risk-exposure ?risk-exposure ?lid ?pid))
;; operator cost
1)

;; Task T7
(:operator (!t7-check-if-acceptable-risk ?lid ?pid ?cid)
;; preconditions
((assign ?ar (is-acceptable-risk ?lid ?pid ?cid))
(assign ?br nil)
(assign ?mr nil))
;; delete list
()
;; add list
((risk-status ?br ?ar ?mr ?lid))
;; operator cost
1)

;; Task T8
(:operator (!t8-check-if-marginally-bad-risk ?lid ?pid ?cid)
;; preconditions
((assign ?mr (is-marginally-bad-risk ?lid ?pid ?cid))
(assign ?br nil)
(assign ?ar nil))
;; delete list
()
;; add list
((risk-status ?br ?ar ?mr ?lid))
;; operator cost
1)

;; Task T9
(:operator (!t9-approve-loan ?lid ?pid ?cid)
;; preconditions
((is-acceptable-risk ?lid ?pid ?cid)
(risk-level ?rl ?lid ?cid ?pid)
(bank-risk-exposure ?risk-exposure ?lid ?pid)
(assign ?loan-status (loan-approval-decision ?lid ?pid ?cid)))
;; delete list
()
;; add list
((loan-status ?loan-status ?lid ?pid ?cid))
;; operator cost
1)

;; Task T10
(:operator (!t10-check-if-bad-risk ?lid ?pid ?cid)
;; preconditions
((assign ?br (is-bad-risk ?lid ?pid ?cid))
(assign ?ar nil)
(assign ?mr nil))
;; delete list
()
;; add list
((risk-status ?br ?ar ?mr ?lid))
;; operator cost
1)

;; Task T11
(:operator (!t11-reject-loan ?lid ?pid ?cid)
;; preconditions
```

```
((is-bad-risk ?lid ?pid ?cid)
(assign ?loan-status (not (loan-rejection-decision ?lid ?pid ?cid))))
;; delete list
()
;; add list
((loan-status ?loan-status ?lid ?pid ?cid))
;; operator cost
1)

;;;*********************************
;;; AXIOMS
;;;*********************************

(:- (applicant-data ?cid)
applicant-data ((customer-id ?cid)))

(:- (credit-history ?cid)
credit-history ((customer-id ?cid)))

(:- (account-data ?cid)
account-data ((customer-id ?cid)))

(:- (property-data ?pid)
property-data ((property-id ?pid)))

(:- (loan-amount ?loan-amt ?lid)
loan-amount ((loan-application-id ?lid) (requested-loan-amount ?loan-amt)))

(:- (is-acceptable-risk ?lid ?pid ?cid)
acceptable-risk ((eval (is-acceptable-risk `?lid `?pid `?cid))))

(:- (is-marginally-bad-risk ?lid ?pid ?cid)
marginally-bad-risk ((eval (is-marginally-bad-risk `?lid `?pid `?cid))))

(:- (is-bad-risk ?lid ?pid ?cid)
bad-risk ((eval (is-bad-risk `?lid `?pid `?cid))))

(:- (calc-credit-rating ?cr-b ?cr-l ?cid)
calc-credit-rating ((eval (calc-credit-rating `?cr-b `?cr-l `?cid))))

;;;*********************************
;;; DEBUGGING METHODS
;;;*********************************

(:method (print-current-state)
((eval (print-current-state)))
())

(:method (print-current-plan)
((eval (print-current-plan)))
())

(:method (print-current-tasks)
((eval (print-current-tasks)))
())

))
```

Appendix C
The LeafHopper ThinkLet Pattern

Identification

Name

LeafHopper

Whats in a name?

A LeafHopper is a small insect that is something like a grasshopper or a cricket. It hops from leaf to leaf eating what it wants, then moving on. We named this thinkLet LeafHopper because the team members can jump from topic-to-topic, contributing as they are inspired, then moving on to new topics.

Script

Overview

All participants view a set of pages, one for each of several discussion topics. Each participant hops among the topics to contribute ideas as dictated by interest and expertise.

Recommended script elements

Do this

→ Assure that participants understand the discussion topics from the `topic_list`.

→ Assure that participants understand the `contribution_prompt`.

→ Explain the mechanics of adding contributions to pages.

Say this

→ "Each of you may have different interests and different expertise."

→ "Start working on the topics in which you have the most interest or the most expertise."

→ "Then, if you have time, move to the other topics to read and comment on the contributions of others."

→ "You may not have time to work on every topic, so work first on the topics that are most important to you."

→ "Read the comments of others and respond if you choose to."

Rules

Role 1: Participant

1. Add any number of contributions to any page.
2. Add only contributions that are relevant to the page topic.
3. Add only contributions that are responsive to the contribution prompt.
4. Shift focus from page to page as interest and inspiration dictate.
5. Read the comments and issues of others.

Parameter

Input parameters

⋆ `Topic_list`: the set of discussion topics

⋆ `Contribution_prompt`: explains what kind of contributions should be added (e.g., problem statements, possible solutions, pros and cons, etc.)

Output parameters

⋆ `Results`: An unfiltered collection of comments organised by topics.

Required capabilities

One page for each topic of discussion, each page labeled with its discussion topic. Participants must be able to see any page at will, must be able to read the contributions of others and must be able to add contributions to any page.

Actions

∗ Add ideas

∗ Read ideas

∗ Choose topics

Selection guidance

PATTERNS OF COLLABORATION

Primary Pattern: Generate

Secondary Pattern: Organize

CHOICE GUIDANCE

Choose this thinkLet

→ When you know in advance that the team must brainstorm on several topics at once.

→ When you want them to generate depth and detail on a focused set of topics.

→ When different participants will have different levels of interest or expertise in the different topics.

→ When it is not important to assure that every participant contributes to every topic.

Do not choose this thinkLet

→ If you want the participants to address topics in a specific order. (Consider DealersChoice instead)

→ If you want the team to breadth and variety. (Consider FreeBrainstorm instead)

→ If you want to assure that all participants to address all issues. (Consider DealersChoice instead)

INSIGHTS ON LEAFHOPPER

Sometimes your team must discuss several topics more or less simultaneously. For example, we have a colleague who worked with a series of groups on resolving pollution issues. He discovered that he got significantly more ideas from a group by posing three simultaneous questions in a LeafHopper ...

What can we do about air pollution?

What can we do about water pollution?

What can we do about ground pollution?

. . . than he got by posing one FreeBrainstorming question or one OnePage question with three parts . . .

What can we do about air, water and ground pollution?

He also got more ideas from the groups by posing the three questions simultaneously than he did by posing them one at a time with a DealersChoice thinkLet. People could hop between the questions as they were inspired. With LeafHopper it is not necessarily the case that every participant will see and contribute to every topic. Sometimes that is exactly why you use it. If you have people with diverse interests, yet you insist that all participants contribute to all topics, some percentage of the group will always be disinterested at any given moment during the activity. If they are allowed to hop at will, all participants can be fully engaged throughout the activity. However, for some kinds of tasks, when it is important that all participants see and contribute to all topics, consider using the DealersChoice thinkLet.

LeafHopper success story

We once worked with a commercial software development team that had 12 tricky issues to resolve. They needed input from engineers, customers, product managers, developers, users and several other success-critical stakeholder groups. They discovered a rare opportunity when all the key stakeholders were to be in the same place at the same time, and managed to schedule a meeting. Then, they realised that although they needed input from all the stakeholders, any given stakeholders only had an interest in about 1/3 of the issues. This meant that no matter what topic was being discussed, 2/3 of these highpowered participants sitting around might be bored. They felt it was impolitic to bore high-powered participants, but unfortunately, the mix of issues and interests was such that they could not

simply schedule subsessions around each topic. We chose a LeafHopper to resolve this dilemma. The development team posted the issues on topic pages in view of the team. They asked the participants to work first on the topics in which they had the most at stake, and on which they had the most expertise. The participants proposed options for resolving each issue, then argued the pros and cons of the proposals. The whole discussion of 12 topics took just over an hour and a half.

In a subsequent BucketWalk thinkLet, the group reached consensus on seven of the outstanding issues and assign action items for collecting information on the other five. The whole group was fully engaged in the activity throughout the event. Said one participant, "We just did a weeks work in three-and-a-half hours."

Combinations

Successors: BucketWalk, StrawPoll

Predecessors: Theme Seeker, OnePage, FastFocus

References

[1] A. Aamodt and E. Plazas. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–52, 1994.

[2] M. Aarup, M. M. Arentoft, Y. Parrod, J. Stader, and I. Stokes. OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraftAIV. In M. Fox and M. Zweben, editors, *Knowledge Based Scheduling*, pages 451–469. Morgan Kaufmann, San Mateo, California, 1994.

[3] A. Abbott. From causes to events: Notes on narrative positivism. *Sociological Methods and Research*, 20(4):428–455, 1992.

[4] K. R. Abbott and S. K. Sarin. Experiences with workflow management: Issues for the next generation. In T. W. Malone, editor, *Proceedings of the 1994 Conference on Computer Supported Cooperative Work*, pages 113–120, Chapel Hill (NC), 1994. ACM.

[5] R. Ackoff. Toward a system of systems concepts. *Management Science*, 17(11):661–671, 1971.

[6] M. Ader. Workflow Comparative Study 2004. http://www.waria.com/books/study-2004.htm, 2004.

[7] J. M. Agosta. Formulation and implementation of an equipment configuration problem withthe SIPE2 generative planner. In *Integrated Planning Applications: Papers from the 1995 AAAI Spring Symposium*, pages 1–10. AAAI Press, Menlo Park, California, 1995.

[8] C. C. Albrecht, D. D. Dean, and J. V. Hansen. Using situation calculus for e-business agents. *Expert Systems with Applications*, 24:391–397, 2003.

[9] H. Alesso and C. F. Smith. *Developing Semantic Web Services*. AK Peters, Ltd., 2004.

[10] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[11] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. F. King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Center for Environmental Structure Series. Oxford University Press, 1977.

[12] R. Allen. Workflow: An introduction. In L. Fischer, editor, *Workflow Handbook 2001*. John Wiley & Sons, New York, NY, 2000.

[13] G. Alonso, D. Agarwal, A. E. Abbadi, and R. Kamath, Mohanand Günthör. Advanced transaction models in workflow contexts. In S. Y. W. Su, editor, *Proceedings of the 12th International Conference on Data Engineering (ICDE-12)*, pages 574–581, New Orleans, LA, 1996. IEEE Computer Society.

[14] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems. *IEEE Expert*, 12(5):xx, 1997.

[15] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. El Abbadi, and C. Mohan. Exotica/FMDC: A workflow management system for mobile and disconnected-clients. *Distributed and Parallel Databases*, 4(3):229–247, July 1996.

[16] G. Alonso and C. Mohan. Workflow management systems: The next generation of distributed processingtools. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures*, pages 32–65. Kluwer Academic Publishers, Boston, MA, 1997.

[17] S. Alter. A taxonomy of decision support systems. *Sloan Managament Review*, 19(1):39–56, Fall 1977.

[18] J. A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 83–88, St. Paul, MN, 1988. AAAI Press.

[19] T. Andrews, F. Curbera, H. Dholakia, Y. G. J. Klein, F. Leymann, K. Liu, D. R. andDoug Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services (BPEL), version 1.1. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel/, May 2003.

[20] S. J. Andriole. The collaborate/integrate business technology strategy. *Communications of the ACM*, 49(5):85–90, 2006.

[21] G. H. Anthes. Sabre flies to open systems. *Computerworld*, 38(22):21–25, May 2004.

[22] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. Cooperative Information Systems. The MIT Press, 2004.

[23] J. H. Appelman and J. van Driel. Crisis-response in the port of rotterdam: can we do without a facilitatorin distributed settings? In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS-38'05)*, Big Island, HI, 2005. IEEE Computer Society.

[24] A. Arkin. Business Process Modeling Language (BPML). http://www.bpmi.org, 2002.

[25] A. Arkin, S. Askary, S. Fordin, W. Jekeli, KohsukeKawaguchi, D. Orchard, S. Pogliani, K. R. andPal Takacsi-Nagy, I. Trickovic, and S. Zimek. W3C Note: Web Services Choreography Interface (WSCI) 1.0. http://www.w3.org/TR/wsci/, August 2002.

[26] S. Arlington, S. Barnett, N. Davies, and J. Palo. Pharma 2010: The threshold of innovation. http://www-1.ibm.com/services/us/index.wss/ibvstudy/imc/a1001099, 2006.

[27] W. R. Ashby. *An Introduction to Cybernetics*. Chapman & Hall Ltd., London, 1956.

[28] P. Avgeriou, D. Vogiatzis, A. Tzanavari, and S. Retalis. Design patterns in adaptive web-based educational systems: An overview. In *Proceedings of the Web Education Conference 2004*, Innsbruck, Austria, February 11–14 2004.

[29] F. Baader, D. Calvanese, D. L. McGuinness, DanieleNardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, UK, 2003.

[30] F. Bacchus. The AIPS '00 planning competition. *AI Magazine*, 22(1):47–56, 2001.

[31] F. Bacchus and M. Ady. Planning with resources and concurrency: A forward chaining approach. In B. Nebel, editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence(IJCAI-01)*, pages 417–424, San Francisco, CA, Aug. 4–10 2001. Morgan Kaufmann Publishers, Inc.

[32] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191, 2000.

[33] C. Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

[34] C. Bäckström and B. Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[35] H. H. Baligh. Decision rules and transactions, organizations and markets. *Management Science*, 32(11):1480–1491, 1986.

[36] H. H. Baligh and R. M. Burton. Describing and designing organizational structures and processes. *International Journal of Policy Analysis and Information Systems*, 5:251–266, 1981.

[37] A. Banerji, C. Bartolini, D. Beringer, VenkateshChopella, K. Govindarajan, A. Karp, H. Kuno, MikeLemon, G. Pogossiants, S. Sharma, and S. Williams. W3C Note: Web Services Conversation Language (WSCL) 1.0. http://www.w3.org/TR/wscl10/, March 2002.

[38] C. Baral and T. C. Son. Extending ConGolog to allow partial ordering. In N. R. Jennings and Y. Lespérance, editors, *Proceedings of the Sixth International Workshop on Agent Theories, Architectures,and Languages (ATAL-99)*, volume 1757 of *Lecture Notes in Computer Science*, pages 188–204. Springer, 1999.

[39] D. Barbará, S. Mehrotra, and M. Rusinkiewicz. INCAs: Managing dynamic workflows in distributed environments. *Journal of Database Management*, 7(1):5–15, 1996.

[40] S. Barley, G. Meyer, and D. Gash. Cultures of culture: Academics, practitioners, and the pragmatics of normativecontrol. *Administrative Science Quarterly*, 31:24–60, 1988.

[41] C. Barnard. *The Functions of the Executive*. Harvard University, Cambridge, MA, 1964.

[42] T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra.* PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998.

[43] A. Basu and R. W. Blanning. A formal approach to workflow analysis. *Information Systems Research*, 11(1):17–36, 2000.

[44] A. Basu and R. W. Blanning. Synthesis and decomposition of processes in organizations. *Information Systems Research*, 14(4):337–355, December 2003.

[45] A. Basu and A. Kumar. Research commentary: Workflow management issues in e–business. *Information Systems Research*, 13(1):1–14, March 2002.

[46] A. Bernstein. How can cooperative work tools support dynamic group process? bridging thespecificity frontier. In *Proceedings of the 2000 ACM conference on Computer supported cooperativework*, pages 279–288. ACM Press, 2000.

[47] A. Bernstein, C. Dellarocas, and M. Klein. Towards adaptive workflow systems: Cscw-98 workshop report. *ACM SIGGROUP Bulletin*, 20(2):54–56, 1999.

[48] K. Bhattacharya, R. Guttman, K. Lyman, F. F. H. III, S. Kumaran, P. Nandi, F. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt. A model-driven approach to industrializing discovery processes in pharamaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.

[49] J. Billington, S. Christensen, K. M. van Hee, EkkartKindler, O. Kummer, L. Petrucci, R. Post, ChristianStehno, and M. Weber. The Petri net markup language: Concepts, technology, and tools. In W. M. P. van der Aalst and E. Best, editors, *Proceedings of the 24th International Conference on Applications and Theoryof Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 483–505. Springer, Eindhoven, The Netherlands, June 23–27 2003.

[50] B. Boehm, P. Grünbacher, and R. O. Briggs. Developing groupware for requirements negotiation: Lessons learned. *IEEE Software*, 18(3):46–55, May-June 2001.

[51] A. J. Bonner. Workflow, transactions and datalog. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposiumon Principles of database systems (PODS '99)*, pages 294–305, New York, NY, USA, 1999. ACM Press.

[52] R. P. Bostrom, R. Anson, and R. Clawson. Group facilitation and group support systems. In L. Jessup and J. Valacich, editors, *Group Support Systems – New Perspectives*. Macmillan, New York, 1993.

[53] BPMI.org. Business Process Modeling Language (BPML). Working Draft 0.4, Business Process Management Initiative, Alameda, CA, [2001–03–08] 2001.

[54] R. J. Brachman and H. J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 2004.

[55] R. O. Briggs. *The Focus Theory of Group Productivity and its Application to Developmentand Testing of Electronic Group Support Systems*. PhD thesis, University of Arizona, Tucson, AZ, 1994.

[56] R. O. Briggs. On theory-driven design of collaboration technology and process. In *[86]*, pages 1–16. Springer, 2004.

[57] R. O. Briggs and G.-J. de Vreede. ThinkLets: Building blocks for concerted collaboration. GroupSystems.com, Tucson, 2001.

[58] R. O. Briggs, G.-J. de Vreede, and J. F. Nunamaker, Jr. Collaboration engineering with thinkLets to pursue sustained sucess withgroup support systems. *Journal of Management Information Systems*, 19(4):31–64, Spring 2003.

[59] R. O. Briggs, G.-J. de Vreede, J. F. Nunamaker, Jr., and DavidTobey. ThinkLets: Achieving predictable, repeatable patterns of group interactionwith group support systems (GSS). In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34'01)*, Maui, HI, 2001. IEEE Computer Society.

[60] J. C. Burns. The evolution of office information systems. *Datamation*, 23(4):60–64, 1977.

[61] S. Carlsen. *Conceptual Modeling and Composition of Flexible Workflow Models*. PhD thesis, Department of Computer Science and Information Science, Norwegian Universityof Science and Technology, Norway, 1997.

[62] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the Troposproject. *Information Systems*, 27(6):365–389, September 2002.

[63] B. Chandrasekaran, T. R. Johnson, and J. W. Smith. Task-structure analysis for knowledge modeling. *Communications of the ACM*, 35(9):124–137, 1992.

[64] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins. What are ontologies and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, Jan/Feb 1999.

[65] J. F. Chang. *Business Process Management Systems: Strategy and Implementation*. Auerbach Publications, 2006.

[66] T. Chapman. Drug discovery: The leading edge. *Nature*, 430:109–115, July 2004.

[67] E. D. Chapple and L. R. Sayles. *The Measure of Management. Designing Organizations for Human Effectiveness*. Macmillan, New York (NY), 1961.

[68] S. A. Chun, V. Atluri, and N. R. Adam. Domain knowledge-based automatic workflow generation. In R. Cicchetti, A. Hameurlain, and R. Traunmüller, editors, *Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA '02)*, volume 2453 of *Lecture Notes in Computer Science*, pages 81–92. Springer Berlin/Heidelberg, Aix-en-Provence, France, September 2002.

[69] A. Cichocki, A. Helal, M. Rusinkiewicz, and DarrellWoelk. *Workflow and Process Automation: Concepts and Technology*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Netherlands, 1998.

[70] V. Clawson, R. Bostrom, and R. Anson. The role of facilitator in computer-supported meetings. *Small Group Research*, 24(4):547–565, 1993.

[71] J. Conklin and M. L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Information Systems (TOIS)*, 6(4):303–331, 1988.

[72] J. D. Couger, M. A. Colter, and R. W. Knapp, editors. *Advanced System Development/Feasibility Techniques*. John Wiley & Sons, New York, 1982.

[73] J. D. Couger and R. W. Knapp, editors. *System Analysis Techniques*. John Wiley & Sons, New York, 1974.

[74] K. Crowston. *Modeling Coordination in Organizations*. PhD thesis, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 1990.

[75] K. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

[76] B. Curtis. Introduction to empirical research on the design process in MCC's software technology program. In *Empirical Studies of the Design Process: Papers for the Second Workshop on Empirical Studies of Programmers*. MCC Technical Report Number STP–260–87, Austin, TX, 1987.

[77] T. H. Davenport. *Process Innovation: Reengineering Work through Information Technology*. Harvard Business School Press, Boston, MA, 1993.

[78] J. Davies, D. Fensel, and F. van Harmelen, editors. *Towards the Semantic Web: Ontology-driven Knowledge Management*. John Wiley and Sons, 2003.

[79] J. Davies, R. Studer, and P. Warren, editors. *Semantic Web Technologies : Trends and Research in Ontology-based Systems*. John Wiley and Sons, 2006.

[80] J. de Graaff. A repeatable process for collaborative knowledge gathering. Master's thesis, Delft University of Technology, The Netherlands, 2004.

[81] G.-J. de Vreede and R. O. Briggs. ThinkLets: Five examples of creating patterns of group interaction. In F. Ackermann and G.-J. de Vreede, editors, *Proceedings of Group Decision and Negotiation 2001 (Joint conference ofthe INFORMS section on Group Decision and Negotiation and the EURO WorkingGroup on Decision Support Systems*, pages 199–208. INFORMS, La Rochelle, France: Faculty of Technology, Policy and Management, DelftUniversity of Technology, June 2001.

[82] G.-J. de Vreede and R. O. Briggs. Collaboration engineering: Designing repeatable processes for high-valuecollaborative tasks. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS-38'05)*, Big Island, HI, 2005. IEEE Computer Society.

[83] G.-J. de Vreede, R. M. Davison, and R. O. Briggs. How a silver bullet may lose its shine. *Communications of the ACM*, 46(8):96–101, 2003.

[84] G.-J. de Vreede and H. de Bruijn. Exploring the boundaries of successful GSS application: Supporting inter-organizationalpolicy networks. *The DATA BASE for Advances in Information Systems*, 30(3-4):111–130, 1999.

[85] G.-J. de Vreede, A. Fruhling, and A. Chakrapani. A repeatable collaboration process for usability testing. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS-38'05)*, Big Island, HI, 2005. IEEE Computer Society.

[86] G.-J. de Vreede, L. A. Guerrero, and G. M. Raventós, editors. *Groupware: Design, Implementation and Use: 10th International Workshop,CRIWG 2004, San Carlos, Costa Rica, September 5–9, 2004. Proceedings*, volume 3198 of *Lecture Notes in Computer Science (LNCS)*. Springer, Berlin/Heidelberg, 2004.

[87] G.-J. de Vreede, G. L. Kolfschoten, and R. O. Briggs. ThinkLets: A collaboration engineering pattern language. *International Journal of Computer Applications in Technology*, 25(2/3):140–154, 2006.

[88] G.-J. de Vreede, D. Vogel, G. Kolfschoten, and JeroenWien. Fifteen years of GSS in the field: A comparison across time and nationalboundaries. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS-36'03)*, Big Island, HI, 2003. IEEE Computer Society.

[89] D. L. Dean and A. V. Deokar. When should group work be engineered? making the collaboration engineeringinvestment decision. *Journal of Management Information Systems*, xx(x):xxx–xxx, 2006.

[90] W. Deiters and V. Gruhn. Process management in practice applying the FUNSOFT net approach to large-scaleprocesses. *Automated Software Engineering*, 5:7–25, 1998.

[91] A. R. Dennis, J. F. George, L. M. Jessup, J. F. Nunamaker, Jr., and D. R. Vogel. Information technology to support electronic meetings. *MIS Quarterly*, 12(4):591–624, December 1988.

[92] A. V. Deokar, T. Madhusudan, R. O. Briggs, and J. F. Nunamaker, Jr. A structured approach to designing interleaved workflow and groupware tasks. In *Proceedings of the Tenth Americas Conference on Information Systems (AMCIS '04)*, New York, NY, August 2004.

[93] G. DeSanctis and R. B. Gallupe. A foundation for the study of group decision support systems. *Management Science*, 33(5):589–609, May 1987.

[94] G. W. Dickson, M. S. Poole, and G. DeSanctis. An overview of the GDSS research project and the SAMM system. In R. P. Bostrom, R. T. Watson, and S. T. Kinney, editors, *Computer Augmented Teamwork: A Guided Tour*. Van Nostrand Reinhold, New York, NY, 1992.

[95] M. Divitini and C. Simone. Supporting different dimensions of adaptability in workflow modeling. *Computer Supported Cooperative Work*, 9(3-4):365–397, 2000.

[96] J. Dobson, A. Blyth, J. Chudge, and R. Strens. *Requirements Engineering: Social and Technical Issues*, chapter The ORDIT Approach to Organisational Requirements, pages 87–106. Academic Press, San Diego, CA, 1994.

[97] P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: Mediating between representation and action in workflow systems. In M. Ackerman, editor, *Proceedings of the 1996 ACM conference on Computer Supported CooperativeWork (CSCW '96)*, pages 190–198, Boston, MA, United States, 1996. ACM Press.

[98] M. Doyle and D. Straus. *How to Make Meeting Work: The New Interaction Method*. Jove Books, New York, NY, 1982.

[99] P. F. Drucker. *The new realities: in government and politics, in economics and business,in society and world view*. Harper & Row, New York, 1989.

[100] DSTC and PrismTech. Submission to Workflow Process Definition: Object Management Group, 2001.

[101] W. Du and A. Elmagarmid. Workflow management: State of the art vs. state of the products. Software Technology Laboratory Report HPL-97-90 HPL-97-90, Hewlett Packard, 1997.

[102] M. Dumas and A. H. ter Hofstede. UML activity diagrams as a workflow specification language. In M. Gogolla and C. Kobryn, editors, *Proceedings of the Fourth International Conference on the Unified Modeling Language (UML '01)*, volume 2185 of *Lecture Notes in Computer Science*, pages 76–90. Springer-Verlag, Toronto, Canada, October 2001.

[103] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, editors. *Process Aware Information Systems: Bridging People and Software throughProcess Technology*. John Wiley & Sons, Inc., 2005.

[104] ebXML Techical Architecture Project Team. ebXML Technical Architecture Specification V. 1.0.4. www.ebxml.org/specs/ebTA.pdf, February 2001.

[105] H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors. *Petri Net Technology for Communication-Based Systems: Advances in Petri Nets*, volume 2472 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin, 2002.

[106] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Conference on Supporting Group Work: Proceedings of Conference on Organizational Computing Systems (COCS '95)*, pages 10–21. ACM Press, 1995.

[107] C. A. Ellis. Information control nets: A mathematical system of office information flow. In *Conference on Simulation, Modeling and Measurement of Computer Systems*, pages 225–240, Boulder, CO, 1979.

[108] C. A. Ellis and M. Bernal. OFFICETALK-D: An experimental office information system. *SIGOA Newsletter*, 3(1):131–140, 1982.

[109] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[110] C. A. Ellis and G. J. Nutt. Office information systems and computer science. *ACM Computing Surveys*, 12(1):27–60, 1980.

[111] B. Enserink. Creating a scenariologic – design and application of a repeatable methodology. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS-36'03)*, Big Island, HI, 2003. IEEE Computer Society.

[112] H.-E. Eriksson and M. Penker. *Business Modeling with UML: Business Patterns at Work*. John Wiley and Sons, 2000.

[113] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence(AAAI-94)*, volume 2, pages 1123–1128, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[114] K. Erol, J. Hendler, and D. S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18:69–93, 1996.

[115] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independentplanning. *Artificial Intelligence*, 76:75–88, 1–2 1995.

[116] R. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, University of Twente, 2002.

[117] R. Eshuis and J. Dehnert. Reactive Petri nets for workflow modeling. In *[377]*, pages 295–314. Springer-Verlag, 2003.

[118] R. Eshuis and R. J. Wieringa. Comparing Petri net and activity diagram variants for workflow modelling– a quest for reactive Petri nets. In *[105]*, pages 321–351. Springer-Verlag, 2002.

[119] R. Espejo, W. Schuhmann, M. Schwaninger, and U. Bilello. *Organizational Transformation and Learning*. Wiley, New York (NY), 1996.

[120] K. J. Fadel and M. Tanniru. A knowledge-centric framework for process redesign. In *SIGMIS CPR '05: Proceedings of the 2005 ACM SIGMIS CPR conference on Computerpersonnel research*, pages 49–58, New York, NY, USA, 2005. ACM Press.

[121] G. Faustmann. Configuration for adaptation – a human-centered approach to flexible workflowenactment. *Computer Supported Cooperative Work*, 9(3-4):413–434, 2000.

[122] H. Fayol. *General and Industrial Management*. Pitman & Sons, London, 1949. (first published in 1916).

[123] P. H. Feiler and W. S. Humprey. Software process development and enactment: concepts and definitions. In *Proceedings of the Second International Conference on Software Process*, pages 28–40. IEEE CS Press, 1993.

[124] FileNet. Business process manager . http://filenet.com/, 2006.

[125] J. Fjermestad and S. R. Hiltz. An assessment of group support systems experimental research: methodologyand results. *Journal of Management Information Systems*, 15(3):7–149, Winter 1998/99.

[126] J. Fjermestad and S. R. Hiltz. Group support systems: A descriptive evaluation of case and field studies. *Journal of Management Information Systems*, 17(3):113–157, Winter 2000.

[127] F. Flores, M. Graves, B. Hartfield, and T. Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, 1988.

[128] M. Fox and D. Long. International planning competition . http://planning.cis.strath.ac.uk/competition/main.html, 2002.

[129] C. Frappaolo. The many generations of workflow. In L. Fischer, editor, *Workflow Handbook 2001*, pages 51–60. Future Strategies, Lighthouse Point (FL), 2000.

[130] P. Freiberger and M. Swaine. *Fire in the valley: The Making of the Personal Computer*. McGraw Hill, New York (NY), 2nd edition edition, 2000.

[131] A. Gabaldon. Programming hierarchical task networks in the situation calculus. In *Proceedings of the AIPS '02 Workshop*. AAAI/IAAI Press, 2002.

[132] M. Gajewski, M. Momotko, H. Meyer, H. Schuschel, and M. Weske. Dynamic failure recovery of generated workflows. In *Proceedings of the Sixteenth International Workshop on Database and Expert Systems Applications (DEXA '05)*, pages 982–986. IEEE, August 22–26 2005.

[133] J. R. Galbraith. *Designing Complex Organizations*. Addison-Wesley, Reading, Mass., 1973.

[134] J. R. Galbraith. *Competing with Flexible Lateral Organizations*. Addison-Wesley, Reading, Mass., second edition, 1994.

[135] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Professional, 1995.

[136] K. Gary, T. Lindquist, H. Koehnemann, and L. Sauer. Automated process support for organizational and personal processes. In *GROUP '97: Proceedings of the international ACM SIGGROUP conference on Supportinggroup work*, pages 221–230, New York, NY, USA, 1997. ACM Press.

[137] M. R. Genesereth. Knowledge Interchange Format - draft proposed American National Standard(dpANS): NCITS.T2/98–004.
http://logic.stanford.edu/kif/dpans.html, 1998.

[138] D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An overview of workflow management: From process modeling to workflow automationinfrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[139] E. M. Gerson and S. L. Star. Analyzing due process in the workplace. *ACM Transactions on Information Systems (TOIS)*, 4(3):257–270, 1986.

[140] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, 2004.

[141] D. Ghosh and M. Tanniru. Generating and evaluating process design options: A network modeling approach. *Annals of Operations Research*, 72:293–299, 1997.

[142] G. M. Giaglis. A taxonomy of business process modeling and information systems modelingtechniques. *International Journal of Flexible Manufacturing Systems*, 13(2):209–228, April 2001.

[143] Y. Gil. Description logics and planning. *AI Magazine*, pages 73–84, Summer 2005.

[144] C. Girault and R. Valk, editors. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, andApplications*. Springer-Verlag, Berlin, 2003.

[145] N. S. Glance, D. S. Pagani, and R. Pareschi. Generalized process structure grammars (GPSG) for flexible representationsof work. In M. Ackerman, editor, *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative-Work (CSCW '96)*, pages 180–189, Boston, MA, 1996. ACM Press.

[146] A. Gomez-Perez, O. Corcho, and M. Fernandez-Lopez. *Ontological Engineering: with examples from the areas of Knowledge Management,e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer, 2004.

[147] P. Graunke, R. B. Findler, S. Krishnamurthi, and M. Felleisen. Modeling Web interactions. In *Lecture Notes in Computer Science: Proceedings of the 12th European Symposium on Programming*, Warsaw, Poland, April 2003. Springer-Verlag.

[148] P. Graunke, R. B. Findler, S. Krishnamurthi, and MatthiasFelleisen. Automatically restructuring programs for the Web. In *Proceedings of the 17th IEEE International Conference on Automated SoftwareEngineering (ASE'01)*, pages 211–222, September 2001.

[149] T. L. Griffith, M. A. Fuller, and G. B. Northcraft. Facilitator influence in group support systems: Intended and unitended effects. *Information Systems Research*, 9(1):20–36, March 1998.

[150] R. Grohowski, C. McGoff, D. Vogel, B. Martz, and J. F. Nunamaker, Jr. Implementing electronic meeting systems at IBM: Lessons learned and successfactors. *MIS Quarterly*, 14(4):369–383, December 1990.

[151] O. M. Group. Uml 1.4 specification. Technical Report formal/01-09-67, Object Management Group, 2001.

[152] V. Grover, S. R. Jeong, W. J. Kettinger, and J. T. C. Teng. The implementation of business process reengineering. *Journal of Management Information Systems*, 12(1):109–144, Summer 1995.

[153] J. Grudin. Why CSCW applications fail: problems in the design and evaluation of organizationof organizational interfaces. In *CSCW '88: Proceedings of the 1988 ACM Conference on Computer–SupportedCooperative Work*, pages 85–93, New York, NY, USA, 1988. ACM Press.

[154] J. Grudin. CSCW: History and focus. *IEEE Computer*, 27(5):19–26, 1994.

[155] J. Grudin and L. Palen. Why groupware succeeds: Discretion or mandate? In *Proceedings of the 4th European Conference on Computer Supported CooperativeWork (ECSCW '95)*, pages 263–278, Stockholm, Sweden, 1995. Kluwer Academic Publishers.

[156] V. Gruhn and W. Deiters. The Funsoft net approach to software process management. *International Journal of Software Engineering and Knowledge Engineering*, 4(2):229–256, 1994.

[157] V. Gruhn and S. Wolf. Software process improvement by business process orientation. *Software Process: Improvement and Practice*, 1(1):49–56, 1995.

[158] M. Grüninger, K. Atefi, and M. S. Fox. Ontologies to support process integration in enterprise engineering. *Computational and Mathematical Organization Theory*, 6:381–394, 2000.

[159] M. Grüninger and C. Menzel. The process specification language (PSL): Theory and applications. *AI Magazine*, pages 63–74, Fall 2003.

[160] L. H. Gulick. Notes on the theory of organizations. In L. H. Gulick and L. F. Urwick, editors, *Papers on the Science of Administration*, page 13. American Management Association, New York, NY, 1937.

[161] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, 1992.

[162] M. Hammer. *Beyond Reengineering: How the Process-Centered Organization is Changingour Work and our Lives*. Harper Business, New York (NY), 1996.

[163] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Harper Business, May 1995.

[164] M. Hammer, W. G. Howe, V. J. Kroksal, and I. Wladawsky. A very high level programming language for data processing applications. *Communications of the ACM*, 20(11):832–840, 1977.

[165] R. J. Harder and H. Higley. Application of thinklets to team cognitive task analysis. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS-37'04)*, Big Island, HI, 2004. IEEE Computer Society.

[166] R. J. Harder and J. M. Keeter. Insights in implementing collaboration engineering. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS-38'05)*, Big Island, HI, 2005. IEEE Computer Society.

[167] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[168] D. Harel. From play-in scenarios to code: An achievable dream. *Computer*, 34(1):53–60, 2001.

[169] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996.

[170] H. J. Harrington. *Business Process Improvement. The Breakthrough Strategy for Total Quality,Productivity, and Competitiveness.* McGraw-Hill, New York, NY, 1991.

[171] N. B. Harrison and J. O. Coplien. Patterns of productive software organizations. *Bell Labs Technical Journal*, 1(1):138–145, 1996.

[172] P. Hart and D. Estrin. Inter-organization computer networks: indications of shifts in interdependence. In *Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office informationsystems*, pages 79–88, New York, NY, USA, 1990. ACM Press.

[173] N. Hayes. Work-arounds and boundary crossing in a high tech optronics company: Therole of co-operative workflow technologies. *Computer Supported Co-operative Work*, 9(3-4):435–455, 2000.

[174] K. Hebbar, S. Smith, I. Minis, and D. S. Nau. Plan-based evaluation of designs for microwave modules. In *Proceedings of the 1996 ASME Design Engineering Technical Conference andComputers in Engineering Conference*, Irvine, CA, August 1996.

[175] S. Henninger and K. Baumgarten. A case-based approach to tailoring software processes. In *Proceedings of ICCBR*, volume 2080 of *Lecture Notes in Artificial Intelligence*, pages 249–262. Springer-Verlag, 2001.

[176] A. R. Hevner and S. T. March. The information systems research cycle. *Computer*, 36(11):111–113, November 2003.

[177] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[178] C. Hewitt. *Artificial intelligence at MIT expanding frontiers*, chapter Organizations are open systems, pages 594–611. MIT Press, Cambridge, MA, USA, 1990.

[179] R. S. Hiltz and M. Turoff. Virtual meetings: Computer conferencing and distributed group support. In R. P. Bostrom, R. T. Watson, and S. T. Kinney, editors, *Computer Augmented Teamwork: A Guided Tour*. Van Nostrand Reinhold, New York, NY, 1992.

[180] V. Hlupic and S. Qureshi. A research model for collaborative value creation from intellectual capital. In *25th International Conference on Information Technology Interfaces (ITI'03)*, pages 431–437, Cavtat, Croatia, June 2003.

[181] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[182] D. Hollingsworth. Workflow Management Coalition The Workflow Reference Model Version 1.1. Document Number WFMC–TC–1003, Workflow Management Coalition, January 19th 1995.

[183] A. W. Holt. Diplans: A new language for the study and implementation of coordination. *ACM Transcations on Information Systems (TOIS)*, 6(2):109–125, 1988.

[184] S. Horn and S. Jablonski. An approach to dynamic adaptation in workflow management applications. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (Workshop: Towards Adaptive Workflow Systems) (CSCW '98)*, page 14, Seattle, November 1998.

[185] IBM Rational Software. Rational software overview. http://www-306.ibm.com/software/rational/, 2006.

[186] IBM, U.K. Business System Development Method, Introducing BSDM, May 1992.

[187] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, London et al., 1996.

[188] I. Jacobson, M. Ericsson, and A. Jacobson. *The Object Advantage: Business Process Reengineering With Object Technology*. Object Technology Series. Addison-Wesley Professional, 1994.

[189] K. Jensen. Coloured petri nets: A high level language for system design and analysis. In G. Rozenberg, editor, *Proceedings of the 10th International Conference on Applications and Theoryof Petri Nets*, volume 483 of *Lecture Notes in Computer Science*, pages 342–416. Springer-Verlag, Berlin, Germany, 1991.

[190] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use:volume 1.* Springer-Verlag, Berlin, Germany, second edition, 1996.

[191] L. M. Jessup and J. Valacich. *Group Support Systems: New Perspectives.* McMillan Publishing Company, New York, NY, 1992.

[192] P. Jonsson and C. Bäckström. State-variable planning under structural restrictions: algorithms and complexity. *Artificial Intelligence*, 100(1-2):125–176, 1998.

[193] S. Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997.

[194] P. J. Kammer, G. A. Bolcer, R. N. Taylor, A. Hitomi, and M. Bergman. Techniques for supporting dynamic and adaptive workflow. *Computer Supported Cooperative Work*, 9(3-4):269–292, 2000.

[195] P. G. Keen and M. S. S. Morton. *Decision Support Systems: An Organizational Perspective.* Addison-Wesley Publishing Company, 1978.

[196] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische prozeßmodellierung auf der grundlage "ereignisgesteuerter prozeßketten (epk)". In A.-W. Scheer, editor, *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, volume 89. Saarbruecken: Universität des Saarlandes, 1992. (in German).

[197] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling inWorkflows.* PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003.

[198] J. Kim and Y. Gil. Knowledge analysis on process models. In *Proceedings of the Seventeenth International Joint Conference on ArtificialIntelligence, IJCAI '01*, pages 935–942, August 4–10 2001.

[199] J. Kim, Y. Gil, and M. Spraragen. A knowledge-based approach to interactive workflow composition. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04), Workshop on Planning and Scheduling for Web and Grid Services*, pages 45–53, Whistler, British Columbia, Canada, 2004.

[200] J. Kim, J. Hahn, and F. J. Lerch. How is the designer different from the user? – focusing on a software developmentmethodology. In *ESP '97: Papers presented at the seventh workshop on Empirical studies ofprogrammers*, pages 69–90, New York, NY, USA, 1997. ACM Press.

[201] J.-H. Kim and C. Huemer. Analysis, transformation and improvements of ebXML choreographies based on workflow patterns. In *Proceedings of the OTM Confederated International Conferences (CoopIS, DOA, and ODBASE)*, volume 3290–3291 of *Lecture Notes in Computer Science*, pages 66–84. Springer-Verlag, Agia Napa, Cyprus, October 2004.

[202] E. Kindler. Using the Petri Net Markup Language for Exchanging BusinessProcesses – Potential and Limitations. In M. Nüttgens and J. Mendling, editors, *XML4BPM 2004, Proceedings of the 1st GI Workshop XML4BPM – XML InterchangeFormats for Business Process Management at 7th GI Conference Modellierung2004, Marburg Germany, March 2004*, pages 43–60, March 2004.

[203] R. Kishore, H. Zhang, and R. Ramesh. A helix-spindle model for ontological engineering. *Communications of the ACM*, 47(2):69–75, 2004.

[204] M. Klein and C. Dellarocas. A knowledge-based approach to handling exceptions in workflow systems. *Computer Supported Cooperative Work*, 9(3-4):399–412, 2000.

[205] A. Knutilla, C. Schlenoff, S. Ray, S. T. Ployak, S. C. Tate, Austinand Cheah, and R. C. Anderson. Process specification language: An analysis of existing representations. NISTIT 6160, National Institute of Standards and Technology (NIST), Gaithersburg, MD, 1998.

[206] J. Koehler, R. Hauser, S. Sendall, and M. Wahler. Declarative techniques for model-driven business process integration. *IBM Systems Journal*, 44(1):47–65, 2005.

[207] G. L. Kolfschoten, J. H. Appelman, R. O. Briggs, and G.-J. Vreede. Recurring patterns of facilitation interventions in GSS sessions. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS-37'04)*, Big Island, HI, 2004. IEEE Computer Society.

[208] G. L. Kolfschoten, R. O. Briggs, J. H. Appelman, and G.-J. Vreede. Thinklets as building blocks for collaboration processes: A further conceptualization. In *[86]*, volume 10.1007/b100403, pages 137–152. Springer, 2004.

[209] G. L. Kolfschoten, R. O. Briggs, G.-J. de Vreede andPeter H.M. Jacobs, and J. H. Appelman. A conceptual foundation of the thinklet concept for collaboration engineering. *International Journal of Human Computer Studies*, 64:611–621, 2006.

[210] G. L. Kolfschoten and W. Veen. Tool support for GSS session design. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS-38'05)*, Big Island, HI, 2005. IEEE Computer Society.

[211] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., September 1993. QA76.76.E95K64 1993.

[212] B. R. Konsynski, J. E. Kottermann, J. F. Nunamaker, Jr., and J. W. Stott. PLEXSYS–84: an integrated development environment for information systems. *Journal of Management Information Systems*, 1(3):64–104, 1984.

[213] B. R. Konsynski and J. F. Nunamaker, Jr. PLEXSYS: a system development system. In Couger et al. [72], pages 399–423.

[214] H. Koontz and C. O'Donnell. *Principles of Management*. McGraw-Hill, New York, NY, 1955.

[215] K. L. Kraemer and J. L. King. Computer-based systems for cooperative work and group decision making. *ACM Computing Surveys (CSUR)*, 20(2):115–146, 1988.

[216] T. Kreifelts, E. Hinrichs, K.-H. Klein, P. Seuffert, and G. Woetzel. Experiences with the DOMINO office procedure system. In L. Bannon, M. Robinson, and K. Schmidt, editors, *Proceedings of the 2nd European Conference on Computer-Supported CooperativeWork (ECSCW '91)*, pages 117–130, Amsterdam, 1993. Kluwer Academic Publishers.

[217] P. Kueng and P. Kawalek. Goal-based business process models: creation and evaluation. *Business Process Management Journal*, 3(1):17–38, April 1997.

[218] H.-U. Küpper. *Controlling. Konzeption, Aufgaben und Instrumente*. Schäffer-Poeschel, Stuttgart, 3rd revised and enhanced edition edition, 2001. (in German).

[219] U. Kuter, E. Sirin, B. Parsia, D. Nau, and J. Hendler. Information gathering during planning for web service composition. *Journal of Web Semantics*, 3(2), 2005.

[220] L. W. Lacy. *OWL: Representing Information Using the Web Ontology Language*. Trafford Publishing, 2005.

[221] K.-Y. Lai, T. W. Malone, and K.-C. Yu. Object lens: a "spreadsheet" for cooperative work. *ACM Transactions on Information Systems (TOIS)*, 6(4):332–353, 1988.

[222] J. Lamb, M. Laskey, and G. Indurkhya. *WebSphere and Lotus: Implementing Collaborative Solutions*. Pearson Education, 2004.

[223] P. R. Lawrence and J. W. Lorsch. *Developing Organizations: Diagnosis and Action*. Addison-Wesley, 1969.

[224] D. B. Leake, editor. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. The AAAI Press/The MIT Press, 1996.

[225] J. Lee. Sibyl: A qualitative decision management system. In P. Winston, editor, *Artificial Intelligence at MIT: Expanding Frontiers*. MIT Press, Cambridge, MA, 1990.

[226] J. Lee, M. Grunninger, Y. Jin, T. Malone, AustinTate, G. Yost, and other members of the PIF Working Group. The PIF process interchange format and framework version 1.2. *The Knowledge Engineering Review*, 13(1):91–120, March 1998.

[227] J. Lee and T. W. Malone. Partially shared views: a scheme for communicating among groups that usedifferent type hierarchies. *ACM Transactions on Information Systems (TOIS)*, 8(1):1–26, 1990.

[228] S. Lemai and F. Ingrand. Interleaving temporal planning and execution: IXTET-EXEC. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI '04)*. AAAI Press, 2004.

[229] Y. Lespérance, H. J. Levesque, and R. Reiter. A situation calculus approach to modeling and programming agents. In M. Wooldridge and A. Rao, editors, *Foundations of Rational Agency*, Applied Logic Series, pages 275–299. Springer, 1999.

[230] S. Letovsky and E. Soloway. Delocalized plans and program comprehension. *IEEE Software*, 3(3):41–49, 1986.

[231] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59– 84, 1997.

[232] P. Levine, J. Clark, C. Casanave, K. Kanaskie, BettyHarvey, J. Clark, N. Smith, J. Yunker, and K. Riemer. ebXML Business Process Specification Schema. http://www.ebxml.org/specs/ebBPSS.pdf, 2001.

[233] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.

[234] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall, Upper Saddle River (NJ), 2000. T58.6.L495 2000.

[235] D. R. Licata and S. Krishnamurthi. Verifying interactive Web programs. In *Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE'04)*, pages 164–173, Linz, Austria, September 2004.

[236] A. Lotem, D. S. Nau, and J. A. Hendler. Using planning graphs for solving HTN planning problems. In *AAAI/IAAI*, pages 534–540, 1999.

[237] P. B. Lowry, C. C. Albrecht, and J. F. a. D. L. Nunamaker, Jr. Evolutionary development and research on Internet-based collaborativewriting tools and processes to enhance eWriting in an eGovernment setting. *Decision Support Systems*, 34(3):229–252, February 2003.

[238] P. B. Lowry, J. F. Nunamaker, Jr., A. Curtis, and M. Lowry. The impact of process structure on novice, internet-based, asynchronous-distributedcollaborative writing teams. *IEEE Transactions on Professional Communication*, 48(4):341–364, 2005.

[239] S. Lukosch and T. Schümmer. Communicating design knowledge with groupware technology patterns: The caseof shared object management. In *[86]*, pages 223–237. Springer, 2004.

[240] A. Macintosh, I. Filby, and A. Tate. Knowledge asset road maps. In *Proceedings of the 2nd International Conference on Practical Aspects ofKnowledge Management (PAKM '98)*, Basel, Switzerland, October 1998.

[241] T. Madhusudan and N. Uttamsingh. A declarative approach to composing web services in dynamic environments. *Decision Support Systems*, 41(2):325–357, January 2006.

[242] T. Madhusudan, J. L. Zhao, and B. Marshall. A case-based reasoning framework for workflow model management. *Data and Knowledge Engineering*, 50(1):87–115, January 2004.

[243] J. Magee and J. Kramer. *Concurrency: State Models & Java Programs*. John Wiley & Sons, 1999.

[244] M. L. Maher and A. G. D. S. Garza. Case-based reasoning in design. *IEEE Expert (Special Issue on AI in Design)*, 12(2):34–41, 1997.

[245] D. Mahling, N. Craven, and W. Croft. From office automation to intelligent workflow systems. *IEEE Expert*, 10(3):41–47, June 1995.

[246] T. W. Malone. Modeling coordination in organizations and markets. *Management Science*, 33(10):1317–1332, 1987.

[247] T. W. Malone. Organizing information processing systems: Parallels between organizations and computer systems. In W. Zachary, S. Robertson, and J. Black, editors, *Cognition, Computation, and Cooperation*, pages 56–83. Ablex, Norwood, NJ, 1990.

[248] T. W. Malone and K. Crowston. What is coordination theory and how can it help design cooperative worksystems? In *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperativework*, pages 357–370, New York, NY, USA, 1990. ACM Press.

[249] T. W. Malone and K. Crowston. Toward an interdisciplinary theory of coordination. Technical report CCS WP 120, Center for Coordination Science, Massachusetts Institute of Technology,Sloan School of Management, 1991.

[250] T. W. Malone, K. Crowston, and G. A. Herman, editors. *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, Cambridge, MA, 2003.

[251] T. W. Malone, K. R. Grant, F. A. Turbak, S. Brobst, and M. D. Cohen. Intelligent information-sharing systems. *Communications of the ACM*, 30(5):390–402, 1987.

[252] J. G. March and H. A. Simon. *Organizations*. Blackwell Publishers, second edition, 1993.

[253] S. T. March, A. R. Hevner, and S. Ram. Research commentary: An agenda for information technology research in heterogeneous and distributed environments. *Information Systems Research*, 11(4):327–341, December 2000.

[254] M. L. Markus and T. Connolly. Why CSCW applications fail: problems in the adoption of interdependentwork tools. In *CSCW '90: Proceedings of the 1990 ACM Conference on Computer–SupportedCooperative Work*, pages 371–380, New York, NY, USA, 1990. ACM Press.

[255] M. L. Markus, A. Majchrzak, and L. Gasser. A design theory for systems that support emergent knowledge processes. *MIS Quarterly*, 26(3):179–212, 2002.

[256] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, editors. *Modelling with Generalized Stochastic Petri Nets*. Series in Parallel Computing. John Wiley & Sons, New York, 1995.

[257] J. Marschak and R. Radner. *Economic Theory of Teams*. Yale University Press, New Haven, 1972.

[258] D. Martin, M. Burstein, J. Hobbs, O. Lassila, DrewMcDermott, S. McIlraith, S. Narayanan, M. P. B. Parsia, T. Payne, E. Sirin, and N. S. K. Sycara. OWL-based Web Service Ontology (OWL-S).
http://www.daml.org/services/owl-s/,
http://www.ai.sri.com/daml/services/owl-s/1.2/, 2006.

[259] G. McCalla and L. Reid. Plan creation, plan execution and knowledge acquisition in a dynamic microworld. *International Journal of Man Machine Studies*, 16:189–208, 1982.

[260] J. McGrath. *Groups: Interaction and Performance*. Prentice-Hall, Englewood Cliffs, N.J., 1984.

[261] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The action workflow approach to workflow management technology. In *Proceedings of the 1992 Conference on Computer Supported Cooperative Work(CSCW '92)*, pages 281–288, New York (NY), 1992. ACM.

[262] J. Mendling and M. Nüttgens. Exchanging EPC business process models with EPML. In M. Nüttgens and J. Mendling, editors, *Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats forBusiness Process Management at 7th GI Conference Modellierung (XML4BPM2004)*, pages 61–80, Marburg, Germany, March 2004.

[263] P. Mi and W. Scacchi. A meta-model for formulating knowledge-based models of software development. *Decision Support Systems*, 17(4):313–330, August 1996.

[264] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[265] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, MA, 1999.

[266] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part i/ii. *Information and Computation*, 100:1–77, 1992.

[267] H. Mintzberg. *The Structuring of Organizations: A Synthesis of the Research*. Englewood Cliffs, N.J.: Prentice Hall, 1979.

[268] H. Mintzberg. *The nature of managerial work*. Prentice Hall College Division, 1980.

[269] L. Mohr. *Explaining Organizational Behavior*. Jossey-Bass, San Francisco, 1982.

[270] L. Morgenstern. Knowledge preconditions for actions and plans. In M. N. Huhns, editor, *Distributed Artificial Intelligence*, pages 192–199. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1988.

[271] S. Mukkamalla and H. Muñoz-Avila. Case acquisition in a project planning environment. In *Proceedings of the Sixth European Conference on CBR*, volume 2416 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2002.

[272] R. Müller, U. Greiner, and E. Rahm. AGENTWORK: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, 51(2):223–256, 2004.

[273] H. Muñoz-Avila, D. W. Aha, and L. Breslow. HICAP: An interactive case-based planning architecture and its applicationto noncombatant evacuation operations. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99);Proceedings of the 11th Conference on Innovative Applications of ArtificialIntelligence*, pages 870–875, Menlo Park, Cal., July 18–22 1999. AAAI/MIT Press.

[274] H. Muñoz-Avila, D. W. Aha, D. S. Nau, R. W. andLen Breslow, and F. Yaman. SiN: Integrating case-based reasoning with task decomposition. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on ArtificialIntelligence (IJCAI-01)*, pages 999–1004, Seattle, Washington, USA, August 2001. Morgan Kaufmann.

[275] H. Muñoz-Avila and F. Weberskirch. Planning for manufacturing workpieces by storing, indexing, and replaying planning decisions. In *Proceedings of the Third International Conference on AI Planning Systems (AIPS '96)*, 1996.

[276] T. Murata. Petri nets: Properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[277] P. Muth, D. Wodtke, J. Weissenfels, A. K. Dittrich, and G. Weikum. From centralized workflow specification to distributed workflow execution. *Journal of Intelligent Information Systems*, 10(2):159–184, 1998.

[278] K. L. Myers, P. A. Jarvis, W. M. Tyson, and M. J. Wolverton. A mixed-initiative framework for robust plan sketching. In *Procedings of the 2003 International Conference on Automated Planning andScheduling (ICAPS '03)*, June 2003.

[279] S. Nambisan. Information sytems as a reference discpline for new product development. *MIS Quarterly*, 27(1):1–18, 2003.

[280] S. Nambisan, R. Agarwal, and M. Tanniru. Organizational mechanisms for enhancing user innovation in information technology. *MIS Quarterly*, 23(3):365–395, 1999.

[281] National Institute of Standard and Technology. Integration Definition for Functional Modeling (IDEF0), December 1993.

[282] D. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem, and StevenMitchell. Total-Order planning with partially ordered subtasks. In B. Nebel, editor, *Proceedings of*

the seventeenth International Conference on Artificial Intelligence(IJCAI-01), pages 425–430, San Francisco, CA, Aug. 4–10 2001. Morgan Kaufmann Publishers, Inc.

[283] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, HectorMuñoz-Avila, J. W. Murdock, D. Wu, and F. Yaman. Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 20(2):34–41, march 2005.

[284] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. WilliamMurdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artifical Intelligence Research*, 20:379–404, December 2003.

[285] D. S. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: Simple hierarchical ordered planner. In D. Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence(IJCAI-99-Vol2)*, pages 968–975, S.F., July 31–Aug. 6 1999. Morgan Kaufmann Publishers.

[286] D. S. Nau, S. J. J. Smith, and K. Erol. Control strategies in HTN planning: Theory versus practice. In *AAAI/IAAI*, pages 1127–1133, 1998.

[287] S. Neumann, C. Probst, and C. Wernsmann. Kontinuierliches prozessmanagement. In J. Becker, M. Kugeler, and M. Rosemann, editors, *Prozessmanagement*, pages 297–323. Berlin et al., 2002.

[288] F. Niedeman, C. M. Beise, and P. M. Beranek. Issues and concerns about computer-supported meetings: The facilitator'sperspective. *MIS Quarterly*, 20(1):1–22, March 1996.

[289] P. Nii. The blackboard model of problem solving. *AI Magazine*, 7(2):38–53, Summer 1986.

[290] I. Nonaka. A dynamic theory of organizational knowledge creation. *Organizational Science*, 5(1):14–37, February 1994.

[291] I. R. Nourbakhsh. *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[292] J. Nunamaker, Jay F. Wits '05 panel: The past, present, and future of design science research. In *In Proceedings of the Fifteenth Annual Workshop on Information Technologies and Systems (WITS '05)*, 2005.

[293] J. F. Nunamaker, Jr. Build and learn, evaluate and learn. *Informatica*, 1(1):1–6, December 1992.

[294] J. F. Nunamaker, Jr. Future research in group support systems: needs, some questions and possibledirections. *International Journal of Human-Computer Studies*, 47:357–385, 1997.

[295] J. F. Nunamaker, Jr., R. O. Briggs, and G.-J. de Vreede. Value creation technology: Changing the focus to the group. In G. W. Dickson and G. DeSanctis, editors, *Information technology and the future enterprise : new models for managers*, chapter 4. Prentice Hall PTR, New Jersey, 2001.

[296] J. F. Nunamaker, Jr., R. O. Briggs, and D. D. Mittleman. Electronic meeting systems: Ten years of lessons learned. In D. Coleman and R. Khanna, editors, *Groupware: Technology and Applications*, chapter 6, pages 149–193. Prentice Hall PTR, Saddle River, NJ, 1995.

[297] J. F. Nunamaker, Jr., R. O. Briggs, D. D. Mittleman, D. Vogel, and P. A. Balthazard. Lessons from a dozen years of group support systems research: A discussionof lab and field findings. *Journal of Management Information Systems*, 13(3):163–207, Winter 1996-97.

[298] J. F. Nunamaker, Jr., R. O. Briggs, and N. C. Romano, Jr. Meeting environments of the future. In *Groupware 1993 Conference Proceedings*, pages 125–143, 1993.

[299] J. F. Nunamaker, Jr., M. Chen, and T. D. Purdin. Systems development in information systems research. *Journal of Management Information Systems*, 7(3):89–106, Winter 1990-91.

[300] J. F. Nunamaker, Jr., A. R. Dennis, J. S. Valacich, D. Vogel, and J. F. George. Electronic meeting systems to support group work. *Communications of the ACM*, 34(7):40–61, 1991.

[301] J. F. Nunamaker, Jr., N. C. Romano, Jr., and R. O. Briggs. Increasing intellectual bandwidth: Generating value from intellectual capitalwith information technology. *Group Decision and Negotiation*, 11(2):69–86, March 2002.

[302] G. J. Nutt. The evolution toward flexible workflow systems. *Distributed Systems Engineering*, 3(4):276–294, December 1996.

[303] G. J. Nutt and C. A. Ellis. Backtalk: an office environment simulator. In *Proceedings of the 1979 International Conference on Communications*, pages 22.3.1–22.3.5, 1979.

[304] Object Management Group. UML 2.0 Superstructure Final Adopted specification . http://www.omg.org/cgi-bin/doc?ptc/2003-08-02.

[305] M. A. Ould. *Business Processes: Modeling and Analysis for Re-engineering and Improvement*. John Wiley and Sons, 1995.

[306] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Fakultät für Mathematik und Physik, Technische Hochschule Darmstadt, Darmstadt, Germany, 1962.

[307] J. Pfeffer. *Organizational Design*. Harlan Davidson, Arlington Heights, III, 1978.

[308] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–361, 1999.

[309] M. E. Pollack. The uses of plans. *Artificial Intelligence*, 57(1):43–68, 1992.

[310] B. Q. Post. A business case framework for group support technology. *Journal of Management Information Systems*, 9(3):7–26, Winter 1992-93.

[311] S. Powers. *Practical RDF*. O'Reilly Media, Inc., 2003.

[312] M. J. Prietula, K. M. Carley, and L. Gasser, editors. *Simulating Organizations: Computational Models of Institutions and Groups*. The MIT Press, Cambridge, MA, 1998.

[313] D. Profozich. *Managing Change with Business Process Simulation*. Prentice Hall, 1998.

[314] F. Puhlmann and M. Weske. Using the π-calculus for formalizing workflow patterns. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and FranciscoCurbera, editors, *Proceedings of the 3rd International Conference on Business Process Management(BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*. Springer Verlag, Nancy, France, September 2005.

[315] B. Ramesh and A. Tiwana. Supporting collaborative process knowledge management in new product developmentteams. *Decision Support Systems*, 27:213–235, 1999.

[316] A. V. Ratzer, L. Wells, H. M. Lassen, M. L. J. F. Qvortrup, M. S. Stissing, M. W. S. Christensen, and K. Jensen. CPN tools for editing, simulating, and analysing coloured Petri nets. In W. M. P. van der Aalst and E. Best, editors, *Proceedings of the 24th International Conference on Applications and Theoryof Petri Nets 2003 (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462. Springer-Verlag, Berlin, June 23–27 2003. http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[317] M. Reichert and P. Dadam. $\text{ADEPT}_{\text{flex}}$ – supporting dynamic changes of workflows without loosing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[318] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 1998.

[319] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Basic Models: Advances in Petri Nets*, volume 1492 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 1998.

[320] R. Reiter. The frame problem in situation the calculus: a simple solution (sometimes)and a completeness result for goal regression. In V. Lefschitz, editor, *Artificial intelligence and mathematical theory of computation: papers inhonor of John McCarthy*, pages 359–380. Academic Press Professional, Inc., San Diego, CA, 1991.

[321] R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64(2):337–351, 1993.

[322] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and ImplementingDynamical Systems*. The MIT Press, 2001.

[323] L. Rising, editor. *Design Patterns in Communication Software*. SIGS Reference Library Series. Cambridge University Press, 2001.

[324] P. Rittgen. Paving the road to business process automation. In *European Conference on Information Systems (ECIS)*, pages 313–319, Vienna, Austria, July 2000.

[325] K. Roberts and G. Gargano. Managing a high reliability organization: A case for interdependence. In *Managing Complexity in High Technology Industries: Systems and People*, pages 147–159. Oxford University Press, New York, 1989.

[326] J. Rockart and J. Short. IT and the networked organization: Toward more effective management ofinterdependence. In *Management in the 1990s Research Program Final Report*. Massachusetts Institute of Technology, Cambridge, MA, 1989.

[327] M. D. Rodriguez-Moreno and P. Kearney. Integrating AI planning techniques with workflow management system. *Knowledge–Based Systems*, 15(5–6):285–291, July 2002.

[328] N. C. Romano, Jr., J. F. Nunamaker, Jr., and R. O. B. D. R. Vogel. Architecture, design, and development of an HTML/JavaScript web-basedgroup support system. *Journal of the American Society for Information Science*, 49(7):649–667, 1998.

[329] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley Longman, 1999.

[330] G. A. Rummler and A. P. Brache. *Improving Performance: How to Manage the White Space on the OrganizationalChart*. Jossey-Bass, May 1995.

[331] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Eduction, Inc., Upper Saddle River, NJ, second edition, 2003.

[332] E. D. Sacerdoti. The nonlinear nature of plans. In *Advance Papers of the Fourth International Joint Conference on ArtificialIntelligence (IJCAI-75)*, pages 206–214, Tbilisi, Georgia, USSR, September 1975.

[333] P. Sachs. Transforming work: collaboration, learning, and design. *Communications of the ACM*, 38(9):36–44, 1995.

[334] K. Salimifard and M. Wright. Petri net-based modelling of workflow systems: An overview. *European Journal of Operational Research*, 134(3):664–676, 2001.

[335] E. Santanen. Resolving ideation paradoxes: Seeing apples as oranges through the clarityof thinklets. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS-38'05)*, Big Island, HI, 2005. IEEE Computer Society.

[336] E. Santanen and G.-J. de Vreede. Creative approaches to measuring creativity: comparing the effectivenessof four divergence thinkLets. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS-37'04)*, pages 29–38, Big Island, HI, January 2004. IEEE Computer Society.

[337] A.-W. Scheer. *ARIS: Business Process Modeling*. Springer, Berlin, second edition, 1999.

[338] C. Schlenoff, G. M., F. Tissot, J. Valois, J. Lubell, and J. Lee. The Process Specification Language (PSL): Overview and Version 1.0 Specification. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, 2000.

[339] G. Schreiber, H. Akkermans, A. Anjewierden, R. deHoogand Nigel Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management: The CommonKADS Methodology*. The MIT Press, 1999.
http://www.commonkads.uva.nl/frameset-commonkads.html.

[340] H. Schuschel and M. Weske. Integrated workflow planning and coordination. In V. Marík, W. Retschitzegger, and O. Stepánková, editors, *Database and Expert Systems Applications, 14th International Conference (DEXA 2003)*, volume 2736 of *Lecture Notes in Computer Science*, pages 771–781. Springer-Verlag Berlin Heidelberg, Prague, Czech Republic, September 1–5 2003.

[341] H. Schuschel and M. Weske. Triggering replanning in an integrated workflow planning and enactment system. In A. Benczúr, J. Demetrovics, and G. Gottlob, editors, *Proceedings of the 8th East European Conference on Advances in Databases and Information Systems (ADBIS 2004)*, volume 3255 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag Berlin Heidelberg, Budapest, Hungary, September 22–25 2004.

[342] J. R. Searle. A taxonomy of illocutionary acts. In *Language, Mind and Knowledge*, pages 344–369. University of Minnesota, Minneapolis, MN, 1975.

[343] M. Shanahan. *Solving the frame problem: A mathematical investigation of the commonsense law of inertia*. MIT Press, Cambridge, MA, USA, 1997.

[344] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[345] M. M. Shepherd, R. O. Briggs, B. A. Reinig, and J. F. Jerome Yenand Nunamaker, Jr. Invoking social comparison to improve electronic brainstorming: Beyond anonymity. *Journal of Management Information Systems*, 12(3):155–170, Winter 1995-96.

[346] A. Sheth. From contemporary workflow process automation to adaptive and dynamic workactivity coordination and collaboration. *ACM SIGGROUP Bulletin*, 18(3):17–20, 1997.

[347] J. Siegel, V. Dubrovsky, S. Kiesler, and T. W. McGuire. Group processes in computer mediated communication. *Organizational Behavior and Human Decision Processes*, 37:157–187, 1986.

[348] H. Simon. *The Sciences of the Artificial*. The MIT Press, third edition, 1996.

[349] H. A. Simon. *Administrative Behavior: A Study of Decision-Making Processes in AdministrativeOrganizations*. New York: Free Press, third edition, 1976.

[350] L. Skyttner. *General Systems Theory*. World Scientific Publishing Company, 2001.

[351] J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1–2):119–153, 2001.

[352] A. Smith. Screening for drug discovery: The leading question. *Nature*, 418:453–459, July 2002.

[353] B. Smith. *Reflection and Semantics in a Procedural Language*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1982.

[354] H. Smith and P. Fingar. *Business Process Management – The Third Wave*. Meghan Kiffer Press, Tampa, FL, 2003.

[355] H. Smith and P. Fingar. Workflow is just a Pi process. *BPTrends*, x:1–36, January 2004.

[356] S. Smith, K. Hebbar, D. S. Nau, and I. Minis. Integrating electrical and mechanical design and process planning. In *Proceedings of the Second IFIP Workshop on Knowledge Intensive CAD*, September 1996.

[357] S. J. J. Smith, D. Nau, and T. Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–106, 1998.

[358] R. H. Sprague. A framework for the development of decision support systems. *MIS Quarterly*, 4(4):1–26, 1980.

[359] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, StanLanning, and L. Suchman. Beyond the chalkboard: computer support for collaboration and problem solvingin meetings. *Communications of the ACM*, 30(1):32–47, January 1987.

[360] E. A. Stohr and J. L. Zhao. Workflow automation: Overview and research issues. *Information Systems Frontier*, 3(3):281–296, 2001.

[361] P. Stone and M. Veloso. User-guided interleaving of planning and execution. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 103–114. IOS Press, Amsterdam, 1996.

[362] L. A. Suchman. *Plans and situated actions: The problem of human-machine communication*. Learning in Doing: Social, Cognitive & Computational Perspectives. Cambridge University Press, New York, NY, USA, 1987.

[363] B. Surjanto, N. Ritter, and H. Loeser. XML content management based on object-relational database technology. In *Proceedings of the First International Conference on Web Information SystemsEngineering (WISE)*, pages 70–79, 2000.

[364] K. D. Swenson and K. Irwin. Workflow technology: Tradeoffs for business process reengineering. In *Proceedings of the Conference on Organizational Computing Systems (COOCS'95)*, pages 22–29, Milpitas (CA), 1995. ACM.

[365] A. Tate. Generating project networks. In R. Reddy, editor, *Proceedings of the Fifth International Conference on Artificial Intelligence(IJCAI-77)*, pages 888–893, Cambridge, MA, August 1977.

[366] F. W. Taylor. *Scientific Management*. Harper & Row, New York (NY), 1947.

[367] S. H. Thomke. *Experimentation matters: unlocking the potential of new techonologies forinnovation*. Harvard Business School Press, 2003.

[368] J. D. Thompson. *Organizations in Action: Social Science Bases of Administrative Theory*. McGraw-Hill, New York, 1967.

[369] J. D. Thompson and M. Zald. *Organizations in Action: Social Science Bases of Administrative Theory*. Transaction Publishers, 2003.

[370] TU Eindhoven and Bakkenist Consultancy, Deloitte & Touche. ExSpect: A software tool embodying a coloured petri nets approach to discreteprocess modelling. http://www.exspect.com/, 2006.

[371] K. T. Ulrich and S. D. Eppinger. *Product Design and Development*. McGraw-Hill/Irwin series in marketing, New York, 3 edition, 2004.

[372] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. AIAI-TR-195, Artificial Intelligence Application Institute, University of Edinburgh, 1998.

[373] J. Valacich, A. Dennis, and J. F. Nunamaker, Jr. Electronic meeting support: the GroupSystems concept. *International Journal of Man-Machine Studies*, 34(2):261–282, 1991.

[374] J. S. Valacich, L. M. Jessup, A. R. Dennis, and J. F. Nunamaker, Jr. A conceptual framework of anonymity in group support systems. *Group Decision and Negotiation*, 1(3):219–241, November 1992.

[375] A. van de Ven. Suggestions for studying strategy process: A research note. *Strategic Management Journal*, 13:169–188, 1992.

[376] A. van de Ven and M. Poole. Methods for studying innovation development in the minnesota innovationresearch program. *Organization Science*, 1:313–335, 1990.

[377] W. van der Aalst and E. Best, editors. *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2003.

[378] W. M. P. van der Aalst. Three good reasons for using a Petri-net-based workflow management system. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC '96)*, pages 179–201, 1996.

[379] W. M. P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[380] W. M. P. van der Aalst. Making work flow: on the application of Petri nets to business process management. In J. Esparza and C. Lakos, editors, *Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, Berlin, Germany, 2002.

[381] W. M. P. van der Aalst. Why workflow is not just a Pi-process. *BPTrends*, x:1–2, February 2004.

[382] W. M. P. van der Aalst. Pi calculus versus Petri nets: Let us eat humble pie rather than furtherinflate the pi hype. *BPTrends*, 3(5):1–11, May 2005.

[383] W. M. P. van der Aalst, A. H. M. T. Hofstede, and B. K. A. P. Barros. Workflow patterns: On the expressive power of (Petri-net-based) workflowlanguages. In K. Jensen, editor, *Proceedings of the Fourth Workshop on the Practical Use of Coloured PetriNets and CPN Tools (CPN 2002)*, volume 560 of *DAIMI*, pages 1–20. University of Aarhus, Aarhus, Denmark, August 2002.

[384] W. M. P. van der Aalst, A. H. M. T. Hofstede, and B. K. A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.

[385] W. M. P. van der Aalst and A. H. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[386] W. M. P. van der Aalst and K. van Hee. *Workflow Management*. MIT Press, 2002.

[387] J. van Grinsven Gert-Jan de Vreede. Addressing productivity concerns in risk management through repeatable distributedcollaboration processes. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS-36'03)*, pages 22–31, Big Island, HI, 2003. IEEE Computer Society.

[388] L. von Bertalanffy. *General System Theory: Foundations, Development, Applications*. George Braziller, 1968.

[389] M. Voorhoeve and W. M. P. van der Aalst. Ad-hoc workflow: problems and solutions. In *Proceedings of the Eighth International Workshop on Database and Expert Systems Applications*, pages 36–40, Toulouse, France, September 1–2 1997.

[390] J. Wainer. Logic representation of processes in work activity coordination. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, pages 203–209, New York, NY, USA, 2000. ACM Press.

[391] M. Weber. *Economy and Society*. University of California Press, Berkeley, CA, 1987.

[392] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 2000.

[393] D. S. Weld. Recent advances in AI planning. *AI Magazine*, pages 93–123, 1999.

[394] C. Welty. Software engineering. In *[29]*, pages 373–387. Cambridge University Press, Cambridge, UK, 2003.

[395] M. Weske. *Workflow Management Systems: Formal Foundation, Conceptual Design, ImplementationAspects*. PhD thesis, University of Muenster, Germany, December 1999.

[396] M. Weske, T. Goesmann, R. Holten, and R. Striemer. Analysing, modelling and improving workflow application development processes. *Software Process Improvement and Practice*, 6(1):35–46, March 2001.

[397] M. Weske, W. M. P. van der Aalst, and H. M. W. Verbeek. Advances in business process management. *Data and Knowledge Engineering*, 50(1):1–8, 2004.

[398] WfMC (Glossary). Terminology and Glossary, 3rd Edition. Document Number WFMC–TC–1011.workflow management coalition, 1999.

[399] WfMC (WPDL). Interface 1: Process Definition Interchange Process Model – WorkflowProcess Definition Language (WPDL). Document Number WFMC–TC–1016–P:Version 1.1. Document Status – Final, 1999.

[400] WfMC (XPDL). Workflow Process Definition Interface – XML Process DefinitionLanguage (XPDL). Document Number WFMC–TC–1025: Version 1.14.Document Status – Final. http://www.wfmc.org/standards/XPDL.htm, October 3 2005.

[401] R. B. White. A prototype for the automated office. *Datamation*, 23(4):83–90, 1977.

[402] S. A. White. Business process modeling notation (bpmn). version 1.0 - may 3, 2004. Technical report, BPMI.org, 2004.

[403] S. A. White. Process modeling notations and workflow patterns. In L. Fischer, editor, *Workflow Handbook 2004*, pages 265–294. Future Strategies, Inc., Lighthouse Point, FL, 2004.

[404] N. Wiener. *Cybernetics*. John Wiley & Sons, New York (NY), 1948.

[405] J. Wiese. *Implementierung der Balanced Scorecard. Grundlagen und IT-Fachkonzept*. Deutscher Universitäts-Verlag, Wiesbaden, 2000. (in German).

[406] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.

[407] E. Winner and M. M. Veloso. Analyzing plans with conditional effects. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proceedings of the Sixth International Conference on Artificial IntelligencePlanning Systems*, pages 23–33, Toulouse, France, April 23–27 2002. AAAI.

[408] T. Winograd. A language/action perspective on the design of cooperative work. In *CSCW '86: Proceedings of the 1986 ACM conference on Computer-supported cooperativework*, pages 203–220, New York, NY, USA, 1986. ACM Press.

[409] T. Winograd. A language/action perspective on the design of cooperative work. *Human Computer Interaction*, 3(1):3–30, 1987–88.

[410] T. Winograd. Designing a new foundation for design. *Communications of the ACM*, 49(5):71–74, 2006.

[411] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New foundation for Design*. Addison-Wesley Professional, 1987.

[412] G. Woetzel and T. Kreifelts. The use of petri nets for modeling workflow in the DOMINO system. In *Proceedings of the Workshop on CSCW, Petri Nets and Related Formalisms atthe International Conference on the Application and Theory of Petri Nets*, pages 11–29, Chicago (IL), 1993.

[413] P. Wohed, W. M. van der Aalst, M. Dumas, and A. H. Hofstede. Analysis of web servives composition languages: The case of BPEL4WS. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER '03)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, 2003.

[414] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. Hofstede, and N. Russell. Pattern-based analysis of UML activity diagrams. BETA Working Paper Series WP 129, Eindhoven University of Technology, Eindhoven, The Netherland, 2004.

[415] E. F. Wolstenholme, S. Henderson, and A. Gavine. *The Evaluation of Management Information Systems: A Dynamic and Holistic Approach*. John Wiley & Sons, Chichester, U.K., 1993.

[416] E. S. Yu, J. Mylopoulos, and Y. Lésperance. AI models for business process reengineering. *IEEE Expert*, pages 16–23, August 1996.

[417] L. Zeng, B. Bentallah, H. Lei, A. H. H. Ngu, D. Flaxer, and H. Chang. Flexible composition of enterprise web services. *Electronic Markets (Special Section: Web Services)*, 13(2):141–152, 2003.

[418] J. L. Zhao and A. Kumar. Data management issues for large scale, distributed workflow systems onthe internet. *ACM SIGMIS Data Base*, 29(4):22–32, 1999.

[419] J. L. Zhao, A. Kumar, and E. A. Stohr. Workflow-centric information distribution through email. *Journal of Management Information Systems*, 17(3):45–72, Winter 2000-01.

[420] M. Zisman. Office automation: Revolution of evolution. *Sloan Management Review*, 19(3):1–16, 1978.

[421] M. zur Muehlen. Organizational management in workflow applications. *Information Technology and Management*, 5(3):271–291, 2004.

[422] M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design, and Applicationof Workflow-driven Process Information Systems*. Advances in Information Systems and Management Science. Logos Verlag Berlin, 2004.

[423] M. zur Muehlen and R. Allen. Stand-alone vs. embedded workflow - putting paradigms in perspective. In L. Fischer, editor, *Excellence in Practice Volume IV: Innovation & Excellence in Workflow and Knowledge Management*, pages 49–58. Future Strategies, Lighthouse Point (FL), 2000.