A LOGIC-BASED METHODOLOGY FOR BUSINESS PROCESS ANALYSIS AND DESIGN: LINKING BUSINESS POLICIES TO WORKFLOW MODELS

by

Harry Jiannan Wang

Copyright © Harry Jiannan Wang 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

2006
As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Harry Jiannan Wang entitled A Logic-Based Methodology for Business Process Analysis and Design: Linking Business Policies to Workflow Models and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

J. Leon Zhao  
Date: 7/25/2006

Jay F. Nunamaker  
Date: 7/25/2006

Mohan Tanniru  
Date: 7/25/2006

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.

Dissertation Director:  J. Leon Zhao  
Date: 7/25/2006
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: _____Harry Jiannan Wang_____

ACKNOWLEDGEMENTS

My thanks and great appreciation first go to my advisor Professor J. Leon Zhao for his guidance, encouragement, patience and kindness over the years. He literally brought me into this academic world, taught me how to be a researcher, helped me reach my full potential, and kept me going when I wanted to give up. I always think I am blessed to have him as my advisor and I do not believe I can finish my PhD without him. Thank you, Dr. Zhao. Thank you for everything.

I am also grateful to my other dissertation committee members, Professor Jay F. Nunamaker and Professor Mohan Tanniru, for their valuable suggestions and feedback on my research and continuous support during my doctoral program.

I must also thank all my friends for always being there for me, sharing their experiences, and offering me their help. In particular, I would like to thank Alan Wang, Lizhi Yang, Jason Li, Wei Chang, Ying Liu, Ming Lin, Aaron Sun, who made these five years much easier and more colorful.
DEDICATION

This dissertation is dedicated

to my wife, Chen Fang,
for always being by my side, and

to my father, Jiachun Wang, my mother, Xiugui Li, and my sister, Yunshu Wang,
for their love and support.
# TABLE OF CONTENTS

LIST OF FIGURES .................................................................................................................. 8

LIST OF TABLES .................................................................................................................... 10

ABSTRACT ................................................................................................................................. 12

1. INTRODUCTION .................................................................................................................. 14

2. LITERATURE REVIEW ........................................................................................................ 19
   2.1 WORKFLOW DEVELOPMENT METHODOLOGY .......................................................... 19
   2.2 PROCESS DESIGN .......................................................................................................... 21
   2.3 PROCESS MODELING AND ANALYSIS ....................................................................... 23
      2.3.1 Control Flow Modeling and Analysis ...................................................................... 24
      2.3.2 Organization and Data flow Modeling ...................................................................... 27
      2.3.3 Workflow Authorization Constraints ....................................................................... 29
   2.4 PROCESS MODELING FOR E-BUSINESS ....................................................................... 32
   2.5 PROCESS CHANGE AND ADAPTIVE WORKFLOW ....................................................... 36

3. A POLICY PERSPECTIVE ON PROCESS ANALYSIS AND DESIGN .................................. 39
   3.1 THE POLICY MISMATCH PROBLEM IN BUSINESS PROCESS MANAGEMENT .......... 41
   3.2 A CASE STUDY OF BUSINESS POLICIES .................................................................... 42
   3.3 PROCESS POLICY TAXONOMY .................................................................................... 45

4. POLICY-DRIVEN PROCESS DESIGN ................................................................................. 51
   4.1 POLICY-DRIVEN PROCESS DESIGN PROCEDURE ....................................................... 51
   4.2 PROCESS POLICY TEMPLATE AND PROCESS MAP CONCEPTS ..................................... 53
   4.3 PROCESS MODEL EXTRACTION PROCEDURE ............................................................. 57
   4.4 VALIDATION OF POLICY-DRIVEN PROCESS DESIGN ................................................ 64
   4.5 ARCHITECTURE OF PPD Prototype System ................................................................. 72
5. A UNIFIED PREDICATE LANGUAGE ......................................................... 79
   5.1 LANGUAGE DEFINITION ........................................................................ 79
   5.2 PROCESS DESIGN VIA UPL ................................................................. 85
   5.3 COMPARISON WITH OTHER LANGUAGES .......................................... 94

6. PROCESS CHANGE ANALYSIS .......................................................... 97
   6.1 A PROCUREMENT APPROVAL PROCESS .......................................... 98
   6.2 PROCESS CHANGE TYPES AND RELATIONSHIP ............................. 103
   6.3 PROCESS CONSTRAINT ANOMALIES ................................................. 106
   6.4 PROCESS CONSTRAINT VERIFICATION ............................................. 109
   6.5 VALIDATION OF THE CONSTRAINT VERIFICATION METHOD ........... 115
   6.6 DESIGN OF A PROCESS CHANGE ANALYSIS TOOL ...................... 118

7. CONCLUSIONS .................................................................................. 121

REFERENCES ........................................................................................... 125
LIST OF FIGURES

Figure 1. Three process design approaches ................................................................. 16
Figure 2. Routing constructs in UML ......................................................................... 25
Figure 3. An example of corporate policies................................................................. 39
Figure 4. Screenshot of an online business policy manual .......................................... 40
Figure 5. The policy mismatch problem in business process management ................. 41
Figure 6. Screenshot of travel policy section in an online manual ............................... 42
Figure 7. Process policy taxonomy ........................................................................... 45
Figure 8. Preliminary travel reimbursement process based on control flow policies ...... 48
Figure 9. Policy-driven process design procedure ..................................................... 52
Figure 10. Detailed process map extraction procedure .............................................. 57
Figure 11. Task view example ................................................................................ 59
Figure 12. Data view example .................................................................................. 61
Figure 13. Draft process map based on the task sequences ....................................... 66
Figure 14. Enhanced process map based on data view ............................................. 68
Figure 15. Revised process map based on PMRules ............................................... 69
Figure 16. Process map after removing semantic errors ......................................... 70
Figure 17. System architecture of PPD tool ............................................................... 73
Figure 18. Screenshot of policy wizard .................................................................... 73
Figure 19. Entity relationship diagram for process policy ....................................... 74
Figure 20. Screenshot of visual editor ...................................................................... 75
Figure 21. Process policy representation stack ....................................................... 76
Figure 22. Procedure for process design via UPL .................................................... 86
LIST OF FIGURES - Continued

Figure 23. Draft control flow diagram based on process policies analysis ...................... 89
Figure 24. Control flow of travel reimbursement process ................................................ 93
Figure 25. Procurement process in UML.......................................................................... 98
Figure 26. Organization model for the procurement process ........................................... 100
Figure 27. Change relationships among process components ........................................... 104
Figure 28. Redesigned procurement process ................................................................. 116
Figure 29. Remodeled organization structure................................................................... 117
Figure 30. Constraint verification algorithm implemented using Jess ......................... 119
LIST OF TABLES

Table 1. Algorithm for drafting process map based on task view .............................. 58
Table 2. Algorithm for process map enhancement based on data view ...................... 60
Table 3. Identified process policies from a business policy manual ............................ 64
Table 4. Formalized travel reimbursement process policies ....................................... 65
Table 5. Data view of travel reimbursement process ................................................... 67
Table 6. Process matrix for travel reimbursement process design .............................. 72
Table 7. Database schema for a policy base ................................................................. 74
Table 8. Process policies represented in RuleML ......................................................... 77
Table 9. Control flow predicates .................................................................................. 80
Table 10. Organization modeling predicates ............................................................... 81
Table 11. Data flow predicates .................................................................................... 82
Table 12. Auxiliary functions ..................................................................................... 83
Table 13. Built-in UPL rules ....................................................................................... 84
Table 14. Travel reimbursement policies ................................................................. 86
Table 15. Identified process elements from process policies ...................................... 87
Table 16. Data dependency rules ............................................................................... 88
Table 17. Data inputs and outputs .............................................................................. 88
Table 18. Partial control flow information .................................................................. 89
Table 19. Algorithm for syntactical verification of control flows ............................... 92
Table 20. Identified design problems ......................................................................... 92
Table 21. Complete control flow information ......................................................... 92
Table 22. Constraints for routing constructs ................................................................. 94
Table 23. Comparisons of three process modeling and analysis approaches ............... 96
Table 24. Descriptions of tasks in procurement process ............................................. 99
Table 25. Control flow of procurement process .......................................................... 99
Table 26. Data items in procurement process ............................................................. 100
Table 27. Organization structure in UPL ................................................................. 100
Table 28. Constraints of requisition approval process .............................................. 101
Table 29. Process Constraints in UPL .................................................................... 102
Table 30. Classification of process changes .............................................................. 103
Table 31. Algorithm for process constraint verification ........................................... 114
Table 32. Changes in the procurement process ....................................................... 115
Table 33. Redesigned procurement process in UPL ................................................. 116
Table 34. New organization structure in UPL .......................................................... 117
ABSTRACT

Today, organizations often need to modify their business processes to cope with changes in the environment, such as mergers/acquisitions, new government regulations, and new customer demand. Most organizations also have a set of business policies defining the way they conduct their business. Although there has been extensive research on process analysis and design, how to systematically extract workflow models from business policies has not been studied, resulting in a missing link between the specification of business policies and the modeling of business processes.

Given that process changes are often determined by executives and managers at the policy level, the aforementioned missing link often leads to inefficient and inaccurate implementation of process changes by business analysts and process designers. We refer to this problem as the policy mismatch problem in business process management. For organizations with large-scale business processes and a large number of business policies, solving the policy mismatch problem is very difficult and challenging.

In this dissertation, we attempt to provide a formal link between business policies and workflow models by proposing a logic-based methodology for process analysis and design. In particular, we first propose a Policy-driven Process Design (PPD) methodology to formalize the procedure of extracting workflow models from business policies. In PPD, narrative process policies are parsed into precise information on various workflow components, and a set of process design rules and algorithms are applied to generate workflow models from that information.

We also develop a logic-based process modeling language named Unified Predicate
Language (UPL). UPL is able to represent all workflow components in a single logic format and provides analytical capability via logic inference and query. We demonstrate UPL’s expressive power and analytical ability by applying it to process design and process change analysis. In particular, we use UPL to define and classify process change anomalies and develop algorithms to verify and enforce process consistency.

The Policy-driven Process Design, Unified Predicate Language, and process change analysis approach found in this dissertation contribute to business process management research by providing a formal methodology for resolving the policy mismatch problem.

**Keywords**: process analysis, process design, process modeling, policy management, predicate logic, workflow management systems
1. INTRODUCTION

Business policies enable the efficient management of an organization by defining the standard procedures for its daily business operations (Peltier 2004). Recently, many organizations are investing a great amount to revamp their business policies in order to comply with new government regulations and quality assurance standards such as Sarbanes-Oxley and ISO9001 (Cobb 2004). For organizations with large-scale business processes, maintaining those processes to meet the changing policies is a non-trivial problem. Therefore, there is an imperative need for a more advanced process analysis and design methodology that can formally link business policies with workflow models to manage their consistency and compatibility in an effective and efficient manner.

Process analysis and design received much attention as a major technique used in business process reengineering initiatives during the 1990s (Hammer et al. 1993). Many process analysis and design projects have successfully enabled organizations to view their business system graphically at any level of detail and complexity and helped them achieve higher level cross-functional collaborations and tangible cost reduction (Madison 2005). However, traditional process analysis and design methods have taken a participative approach which can be resource-intensive and time-consuming, involving extensive meetings, workshops, and interviews to collect data on business processes (Hunt 1996; Reijers et al. 2003). Different from the participative approach, an analytical approach aims to derive and analyze workflow models by applying formal theories and algorithms (Aldowaisan et al. 1999; Datta 1998; Hofacker et al. 2001; Reijers et al. 2003;
van der Aalst 2000a). However, very few analytical approaches have been implemented in real process analysis and design projects due to their restricted assumptions and steep learning curve (Wang et al. 2005). Furthermore, none of the existing process analysis and design methods explicitly investigate the relationship between business policies and workflow models, resulting in a missing link between the specification of business policies and the modeling of business processes.

In this dissertation, we propose an innovative process analysis and design procedure that leverages detailed business policies to build workflow models, which we refer to as Policy-driven Process Design (PPD). In PPD, narrative process policies are first formalized into precise information on different workflow components, such as control flow, data flow, organizational model, and process constraints. Then, a set of process design rules and algorithms are developed to systematically extract workflow models from those formalized process policies. As shown in Figure 1, PPD advocates a new way of discovering processes different from participative and analytical approaches (Cobb 2004; Datta 1998; Hunt 1996; Madison 2005; Reijers et al. 2003; Weske et al. 1999). Specifically, PPD minimizes the time-consuming and ambiguous data collection used in participative approaches by systematically analyzing existing business policies to extract useful process information. It is also more pragmatic than analytical approaches because it provides detailed instructions on process extraction procedure via intuitive design rules and algorithms. Furthermore, PPD builds the formal linkage between business policies and workflow models, which facilitates process change propagation from high-level organizational policies to low-level operational processes and enables systematic,
efficient and accurate policy-process compatibility maintenance.

We demonstrate the feasibility of Policy-driven Process Design methodology by designing and developing two process design tools, namely, Policy Wizard and Visual Editor. In particular, policy wizard are implemented as a set of dynamic webpages based on PPD rules and algorithms. Policy wizard provides step-by-step instructions to help process designers parse narrative business policies into precise information on different process components. The visual editor can visualize control flow as a directed graph and assist process designers to identify potential process design errors.

![Three process design approaches](image)

**Figure 1. Three process design approaches**

In today’s dynamic business environment, companies need to modify their business processes in response to new customer demand and market opportunities. Without proper analysis and management, dynamic process changes can lead to costly runtime workflow errors. For instance, mergers/acquisitions may result in removing organizational roles. Then, the tasks assigned to those roles must be delegated to other resources, otherwise a runtime role resolution error will occur and the tasks will never be executed. Given that a
workflow model consists of four major components, namely, control flow, data flow, organizational model, and process constraints, changes in one component may have significant impact on other components (Basu et al. 2000; Wang et al. 2004). Although many process modeling languages have been proposed, such as UML (OMG 2003), Petri nets (Adam et al. 1998; van der Aalst 1998), and Metagraphs (Basu et al. 2000; Basu et al. 2003), different workflow components are usually modeled in different notations, which makes the change analysis among those components very difficult. As well as being a new process design methodology, our proposed Policy-driven Process Design also provides a unified policy view of different workflow components, which leads us to the development of a logic-based process modeling language named Unified Predicate Language (UPL).

UPL is grounded in predicate logic and consists of a set of predicates, functions, and rules. UPL is expressive yet simple enough to represent different workflow components in a single logical format, and it also has rich analytical capabilities based on logic inferences and queries. We demonstrate the power of UPL by applying it in process design and process change analysis. In particular, UPL is able to provide control flow design insights by reasoning about data flow and verify process design syntactical correctness by conducting logic queries.

Then, we investigate process changes and classify them into three basic operations on different workflow components, namely, insertion, deletion, and update. After discussing different types of process change relationships, we study how changes in control flow and organizational model can affect process constraints. We use UPL to formally define four process constraint change anomalies, i.e., missing constraint, redundant constraint, invalid
constraint, and conflicting constraint. We prove that a set of process constraints is consistent if those four process constraint anomalies do not exist. An algorithm is developed to verify and enforce process constraint consistency. A Java-based rule engine named Jess is used to provide the running environment for logic-based process change analysis. In particular, Unified Predicate Language is mapped into the Jess language and the constraint verification algorithm is implemented as Jess programs. UPL is also compared with two other well-know process modeling languages: Petri nets and Metagraphs. The comparison result shows that UPL has several advantages over Petri nets and Metagraphs, especially in unified process modeling and capability in process design and change analysis.

The rest of the dissertation proceeds as follows. We first review the related literature in Section 2. Then we present a new policy perspective on process design and analysis in Section 3. We show that rich information on process models can be found in business policies, and workflow models can be retrieved from those policies with the help of a process policy taxonomy. In Section 4, we discuss our innovative policy-driven process design methodology. The Unified Predicate Language is defined and applied in process design in Section 5, and process change analysis is conducted in Section 6. We conclude this dissertation in Section 7 by highlighting our contributions and outlining a future research plan.
2. LITERATURE REVIEW

In this section, we review related literature in three major research areas in business process management: workflow development methodology, process modeling and analysis, and adaptive workflow.

2.1 Workflow Development Methodology

Although workflow management systems have been deployed in many organizations, most workflow development projects are still conducted on an ad hoc basis resulting in inefficient and low quality process design and workflow applications. Several workflow development methodologies and reference models have been proposed to provide guidelines to process analysts and designers. A workflow-aware information system development methodology named WISDM was presented in (Kwan et al. 1998). Five key phases in workflow development are identified, namely, study phase, analysis phase, design phase, coding, testing and documentation phase, and implementation and evaluation phase. A set of process modeling techniques, such as data flow diagram, entity relationship diagram, IDEF3, PERT, and flow chart, are compared in terms of their capabilities to model different perspectives of workflow model. Another reference model for workflow application development processes is discussed in (Weske et al. 1999). Compared with WISDM, this model provides more details on each step of a workflow development process. For instance, for the survey phase, the authors includes detailed information on various activities that should be conducted, such as interviews, group
meetings, organization structure analysis, and review meetings, etc. Several case studies are also presented to illustrate the reference model.

Bajaj and Sudha (2002) proposed a State-Entity-Activity-Model (SEAM) workflow development methodology (Bajaj et al. 2002). SEAM emphasizes on providing a single view of different aspects of workflow model including data, control flow and organization by defining a set of workflow modeling constructs and schemes. SEAM is also capable to map the conceptual workflow model into relational database systems to facilitate workflow application development. ARIS (Architecture of Integrated Information Systems) is another well-known process development methodology (Scheer 2000a). It offers a framework to completely describe business processes, integrates the most suitable methods for process modeling, and provides reference models as tools for administrating workflow applications (Scheer 2000b). ARIS has been widely adopted by industry practitioners and has been incorporated in several commercial product offerings from companies like SAP and IDS Scheer.

Different from the approaches aforementioned, we study workflow development from a unique policy perspective, where different process components, such as control flow, data flow, organizational model, and constraints are represented as a set of process policies and workflow models can be constructed by systematically analyzing those policies. We will illustrate that this policy-driven process development methodology can greatly improve the efficiency of process design and reduce development cost. In addition, although analyzing and designing AS-IS process model is identified as a major step in all methodologies aforementioned, the procedure is only discussed at a very high level and
tends to be ad hoc. The our proposed methodology fills this void in workflow development methodology research by providing detailed and systematic guidelines for extracting workflow models from business policies.

2.2 Process Design

Since early 90s, Business Process Reengineering (BPR) has attracted much attention in both industry and academia. As a critical stage of BPR, identifying and analyzing existing organizational processes is the foundation for any further process changes and improvements (Hammer et al. 1993; Kettinger et al. 1997). The goal of process design projects is to help organizations identify, understand, and improve their AS-IS processes by applying process mapping methodologies and related tools (Hunt 1996). Most process design projects are mainly conducted in a participative manner, where process information is obtained via extensive interviews, meetings, and workshops (Cobb 2004; Kettinger et al. 1997; Madison 2005). Although participative process mapping projects can collect the most detailed process information, they tend to be time-consuming and resource-intensive. Moreover, due to their subjective nature and incorporation of different user opinions, the collected data usually contains ambiguous, uncertain, and conflicting information, which makes process model extractions from those data very difficult (Reijers et al. 2003).

Computer-Aided techniques for structured documentation and analysis of information systems have been proposed to improve manual documentation (Nunamaker 1971; Teichroew et al. 1977; Teichroew et al. 1979). In particular, PSL (Problem
Statement Language)/PSA (Problem Statement Analyzer) approach has been developed to express system requirements in an unambiguous machine-processable form, which can be checked for correctness and consistency by computers (Teichroew 1970). PSL is designed with precise syntax and semantics and system requirements are represented as objects and their relationships using predefined object and relationship names. Once system statement is specified in PSL format, PSA can performs numerous checks and analyses and generates warnings, diagnostics, and reports (Teichroew et al. 1979). However, due to a series of problems including high license fee, restricted documentations in public domain, and obsolete technologies, PSL/PSA is not widely adopted nowadays (www.pslpsa.com).

Different from participative approaches, an analytical process design approach aims to derive process model by using formal theories and techniques. Various formal methods such as linear programming (Aldowaisan et al. 1999), cost optimization (van der Aalst 2000a), computational experiments (Hofacker et al. 2001), probability theory (Datta 1998) have been applied to analytical process design and redesign. Reijers et al. (2003) proposed a product-based workflow design (PBWD) method. PBWD takes the structure and characteristics of information intensive products, such as bill of material and insurance plan, as the major workflow design inputs and derives a favorable design of the process based on the analysis of data dependency, task execution sequences, product rules, and three process design criteria, namely, quality, costs and time. Another workflow design method based on data flow analysis is presented in (Sun et al. 2004a). Intuitively, if a task $t_1$ outputs some data items that are inputs for a task $t_2$, then $t_1$ should be executed...
and if there are no data dependencies among two tasks, they can be executed in parallel without creating any data flow errors. Although analytical process design approaches provide theoretical foundation for the development of advanced process analysis tools, the learning curve for those methods are very steep due to their mathematical nature resulting in very few implementations in real process design projects.

In this dissertation, we propose a policy-driven process design, which incorporates features from both participative and analytical approaches and tries to minimize their drawbacks by leveraging systematic process policy analysis. In addition, none of the existing approaches aforementioned explicitly leverages business policies in the process design procedure, which often leads to the incompatibility between organizational policies and process models. Our approach builds the formal linkage between business policies and processes, which makes policy-process compatibility maintenance more systematic, efficient and accurate.

2.3 Process Modeling and Analysis

A business process is defined as “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” by Workflow Management Coalition (WfMC 1999b). Aligned with WfMC’s business process definition, a process model can be studied from five perspectives, namely, behavioral, informational, organizational, functional, and operational perspectives (Curtis et al. 1992;
Jablonski et al. 1996; Stohr et al. 2001). The behavioral perspective specifies the relationships among tasks, agents, roles and data. Specifically, it defines the execution order of tasks such as sequence, parallelism, split, and iteration, which is often referred to as control flow. In addition, constraints or rules are defined to specify how agents are associated with roles, how roles are assigned to tasks, and what data are related to each task. The informational perspective describes data and information that are consumed, produced and transferred by tasks. The organizational perspective defines the organizational hierarchy of users, roles, and organization units. The functional perspective describes functions and outcomes accomplished by activities and the operational perspective focuses on tools and applications that carry out the execution of tasks in a process model.

Because functional and operational perspectives emphasize more on the implementation and system, most existing research on process modeling and analysis focus on the behavioral, informational, and organizational perspectives (Basu et al. 2002). Specifically, control flow, data flow, organizational model, and process constraints are the key components of a process model, which have been extensively studied in the literature as we discuss next.

2.3.1 Control Flow Modeling and Analysis

The structural aspect of business process modeling is captured by control flow (Stohr et al. 2001). Control flow usually specifies tasks, their execution sequences, and various routing conditions (WfMC 1999b). There has been extensive research effort on control
flow modeling, resulting in many process modeling methods, such as Unified Modeling Language (UML) (OMG 2003), Petri Nets (Murata 1989; van der Aalst 1998), Metagraphs (Basu et al. 1999; Basu et al. 2000; Basu et al. 2003), and logic-based languages (Davulcu et al. 1998). The activity diagram in UML is one of the mostly adopted business process modeling standards. In terms of control flow, it supports five basic routing constructs as shown in Figure 2 (WfMC 1999b).

Sequence means that tasks are executed sequentially if the execution of one task is followed by the next task as shown in Figure 2(a). Fork and Join are used to model parallel task execution. In Figure 2(b), after the execution of t₁, t₂ and t₃ are activated at the same time. In Figure 2(c), only after both t₁ and t₂ complete execution, t₃ can be triggered. Branch and Merge are used to model conditional routing. In Figure 2(d), after t₁, either t₂ or t₃ but not both are executed. In Figure 2(e), any one of t₁ and t₂ finishes execution, t₃ is executed. These five basic routing constructs can model most common business processes, whereas more thorough and complicated study on control flow
structure can be found in research about workflow patterns (van der Aalst et al. 2003b). However, activity diagram lacks of rigorous mathematical foundation and therefore has little analytical capability.

As a graphical and mathematical modeling tool, Petri nets have been used to represent and analyze process models (Adam et al. 1998; Murata 1989; van der Aalst 1998). A Petri net consists of three main components: transition, place, and directed arc. Transitions represent activities or tasks; places specify different routing conditions, and directed arcs are execution path linking transitions and places. In addition, places can hold tokens, which represent the states of workflow execution. The main advantages of Petri nets formalism include a formal theory base, a token-based representation of workflow states, and its rigorous analysis and verification tools. Metagraphs are another rigorous process modeling tool with a mathematical foundation and strong analytical capability. Metagraphs provide three different views of a process model, namely, task view, data view and resources view. The interactions among those three views can be analyzed via matrix computations (Basu et al. 1995; Basu et al. 1999; Basu et al. 2000; Basu et al. 2003). Various logic formalisms have also been applied to process modeling and analysis, such as temporal logic, event algebra, and concurrent transaction logic (Davulcu et al. 1998; Mukherjee et al. 2003). One unique feature of logic-based approaches is the capability to model and enforce various constraints on task dependency and execution orders.

In this dissertation, we propose a process modeling language based on predicate logic. Our language leverages concepts in UML to model basic control flow structure, which
make it succinct yet expressive. We also incorporate research in logic-based workflow modeling to conduct process analysis via logic inference and queries. More details on our proposed language will be discussed in Section 4.

2.3.2 Organization and Data flow Modeling

Organizational modeling in workflow has been identified as an important research area in business process management, which provides the organizational context of workflow application (Basu et al. 2002). An organizational model represents the organization structure and related business rules, where organization structure is usually specified in the form of role hierarchy and business rules are translated into process constraints. Role-based access control model is proposed to simplify management of authorization and provide greater flexibility in specifying and enforcing security policies (Ferralolo et al. 2001; Sandhu et al. 1996). Roles represent business functions in a given organization, such as CEO, purchasing manager, buyer, etc. Authorizations to execute tasks are granted to roles rather than individual users. Moreover, roles are usually structured into role hierarchies to reflect organization’s lines of authority and responsibility (Sandhu et al. 1996). Organizations also have policies in place defining what tasks can be executed by which users. More sophisticated security policies such as separation of duties, binding of duties are also specified to maintain process integrity. Although organizational modeling is very important for proper workflow execution, most existing workflow management systems focus on control flow modeling and oversimplify the organizational dimension (Basu et al. 2002).
Several organizational meta models have been proposed to make workflow management systems more “organizational aware” (Basu et al. 2002). A reference organization model has been issued by Workflow Management Coalition, which defines a list of workflow participants and their relationship (WfMC 1999a). In particular, four types of workflow participants are identified, namely, human (user), role, organization unit, and resource (program or machine agent). A more complicated organization model is presented in (van der Aalst et al. 2003a), which includes additional features like availability of resources, delegation and role inheritance. Their organization model is specified in UML and then converted into an XML DTD, which is used by eXchangeable Routing Language (XRL) to exchange organization information in cross-organizational processes (van der Aalst et al. 2003a). Another organization meta-model is proposed as a benchmark to evaluate different workflow management systems (zur Muhlen 1999a). Besides defining workflow participant types and their relationships, another important aspect of organizational modeling is the specification of policies and patterns that define how to assign different workflow participants to various tasks. Some basic assignment and synchronization policies are discussed in (zur Muhlen 1999b; zur Muhlen 2004). As we will show in section 4, we leverage the research in organizational modeling to design organizational predicates in our proposed logic-based language for process analysis and design.

The modeling, specification, and validation of data flow in workflow system is also an importation research area, because potential data flow problems may prevent the process from proper execution if not detected prior to workflow deployment (Sadiq et al. 2004; Sun
et al. 2004b). Specifically, seven data validation problems are identified as redundant data, lost data, missing data, mismatched data, inconsistent data, misdirected data, and insufficient data (Sadiq et al. 2004). A set of validation algorithms are designed to identify those data flow anomalies (Sun et al. 2004b). Recent research on workflow data patterns aims to capture the various ways in which data is represented and utilized in workflow (Russell et al. 2005). For instance, there are basically two data allocation mechanisms, namely, push and pull (Russell et al. 2005; zur Muhlen 2004). In the push approach, the business process management system allocates the work items to workers. In contrast, in the pull approach, the workers can take the initiative to allocate or start a work item once it is made available by the system. Analysis on data dependency can also facilitate process design by validate the correctness of activity sequencing (Reijers et al. 2003; Sun et al. 2004a). We leverage the concepts and techniques developed in data flow research to design data predicates in our proposed language as discussed later in section 4.

2.3.3 Workflow Authorization Constraints

As we discussed in previous sections, process design involves modeling different aspects of business processes, among which workflow security modeling has been identified as an important issue (Basu et al. 2002; Stohr et al. 2001). In particular, the proper execution of processes requires authorization constraints to enforce the assignment of tasks to organizational resources, such as human users, roles, organization units, or machine agents (Casati et al. 2001; Casati et al. 1999). Traditional access control models, such as mandatory and discretionary security model (Atluri et al. 1997), role-based access
control (Sandhu et al. 1996), task-based access control (Thomas 1997; Thomas et al. 1997), have been applied to workflow management (Wu et al. 2002).

Ferraiolo et al. (2001) proposed role-based access control (RBAC) model to simplify management of authorization while providing an opportunity for greater flexibility in specifying and enforcing enterprise-specific security policies. In RBAC model, roles represent business functions in a given organization, such as CEO, purchasing manager, buyer, etc. Authorizations are then granted to roles, rather than users. The authorizations granted to a role are strictly related to the data objects and resources that are needed for executing the functions associated with that role (Ferralolo et al. 2001). RBAC has been a basic access control model supported by most existing workflow management systems. Workflow systems also have more specific authorization requirements that cannot be handled by RBAC. For instance, Task-based Access Control (TBAC) uses tasks as an important parameter for access control and authorization (Thomas 1997; Thomas et al. 1997). It is an active security model that is well suited for information processing activities, where users access data and applications in order to perform certain tasks. TBAC approaches security management from an application perspective rather than from a system-centric subject-object view. In the subject-object paradigm, access decision function checks whether a subject has the required permissions for the operation, but it does not take into account the context of the access. In addition, TBAC paradigm also considers the temporal constraints where access is permitted based on a just-in-time fashion for the activities or tasks in consideration (Thomas et al. 1997). By incorporating these traditional access control models into workflow management, a set of workflow
authorization models have been proposed (Atluri et al. 1996a; Atluri et al. 2000; Huang et al. 1998).

Workflow authorization constraints can be classified into two categories based on their verification time: static vs. dynamic authorization constraints (Bertino et al. 1999; Casati et al. 1998). Static authorization constraints can be evaluated without executing the workflow, while dynamic constraints can only be evaluated during workflow execution because they require the comparison between current and previous states of the workflow (Casati et al. 1998). Many types of workflow authorization constraints have been identified and summarized in (Casati et al. 1998). In particular, separation of duties and binding of duties are two well-known dynamic constraints. Separation of duties constraint is designed to reduce the risk of fraud by limiting any individual’s authority to commit fraud on his own. For instance, an example can be “a purchase request cannot be approved by the same person who submitted it”. Binding of duties constraint aims to improve efficiency by grouping a set of activities and assigning them to the same resource. For example, be that “the same representative who approves a customer’s credit card application should also handle all that customer’s questions if any” is an instance of binding of duties constraint.

Different mechanisms have been proposed to specify and enforce workflow authorization constraints. Atluri and Huang (1996) extended Petri nets by proposing a Secure Petri Net (SPN) to specify multi-level security constraints. Then, SPN is used to detect and prevent task dependencies that violate those constraints (Atluri et al. 1996b). Bertino et al. (1999) presented a logic-based framework for the specification and enforcement of workflow authorization constraints. Their work focused on the automatic
role and user assignment to tasks without violating any predefined constraints (Bertino et al. 1999). In particular, a constraint specification language is proposed to express both static and dynamic authorization constraints as logic clauses and formal definitions on constraint consistency were provided. They also developed a set of algorithms to verify constraint consistency and generate different plans on valid user-task assignments. Casati et al. (2001) studied workflow authorization constraints using active database technology. Process constraints are described as ECA (event-condition-action) rules based on workflow instance, time, and history (Casati et al. 2001). A set of patterns for ECA rule design is presented in (Casati et al. 2000).

However, none of those research efforts has thus far addressed the dynamic change problem of workflow constraints. How process changes in control flow, organizational model, and data flow can affect the consistency of workflow authorization constraints has not been studied in the literature. In this dissertation, our proposed logic-based language borrows ideas from existing logic-based constraint specification languages and extends them to model all basic workflow routing constructs and complex organizational model. In addition, we formally define workflow constraint consistency under various process changes and develop algorithms to verify constraint consistency and detect potential constraint anomalies.

2.4 Process Modeling for E-business

In order to offer easy integration with a wide range of applications, many companies are investing considerable resources to expose their business operations with web services
technology (Geer 2003). Built on existing and emerging standards such as HTTP, Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI), web services allow business functions from different companies to be loosely coupled more rapidly and less expensively (Oellermann 2001). Furthermore, a higher business value can be achieved if individual web services in a supply chain can be connected to form composite business processes (Leymann et al. 2002).

As an open and standards-based approach for web services connectivity, web services orchestration is receiving increasing attention. This standard describes how web services can interact with one another at the message level and defines the business logic and execution order of web services interactions (Peltz 2003). It also assumes that one of the business parties central controls the process, which makes the process management easier. Web services orchestration provides an open, standard-based approach for connecting web services together to create higher-level business processes (Peltz 2003). Standards have been developed to reduce the complexity required to orchestrate web services. Some early work in this standard development includes XLANG from Microsoft and Web Services Flow Language (WSFL) from IBM. Recently, Business Process Execution Language (BPEL) was jointly released by IBM, Microsoft, and BEA as a combination of the best standards for web services composition, such as XLANG and WSFL (Andrews et al. 2003). BPEL is built on top of Web Service Definition Language (WSDL), which is used to describe the entry and exit points of BPEL processes, the external services and the data type of information exchanged between requests (Peltz 2003). BPEL includes support for
both basic and structured activities. Basic activities are in charge of receiving or replying to message requests as well as invoking external services, whereas structured activities manage the overall process flow, specifying what activities should run and in what order. BPEL also supports conditional looping and dynamic branching using <switch>, <while>, and <pick> tags. The rich set of elements makes BPEL very expressive in terms of control flow modeling, which has been discussed from workflow pattern perspective in (Wohed et al. 2002). BPEL has received remarkable attention and moved into a number of product implementations, such as IBM’s WebSphere Process Server© and Oracle’s BPEL Process Manager©.

Despite the promise, the absence of security is proving to be a major obstacle in convincing companies to participate in web services orchestration. As such, there have been lots of research efforts in the development of web services security, and several security specifications have been proposed, such as WS-Security, Extensible Access Control Markup Language (XACML), and Security Assertion Markup Language (SAML) (Naedele 2003). WS-Security can ensure that messages are not modified or forged while in transit or while residing at destinations. Extensible Access Control Markup Language (XACML) is an XML based access control policy specification language. It was developed by OASIS under the sponsorship of several major industry vendors, such as IBM, SUN, BEA and Entrust (OASIS 2003). This policy language is used to describe access control requirements in any environments, form a query to ask whether or not a given action should be allowed, and interpret the result. Although web services technology facilitates inter-organizational computing and integration, there has been no standard on how to share
security information across different security domains. Security Assertion Makeup Language (SAML) is used to communicate the authorization information among different companies (OASIS, 2002). In SAML, the security information exchanged is in the form of an assertion about subjects. There are three types of core assertions defined by SAML, namely, authentication assertion, attribute assertion, and authorization assertion. With these technologies, various authorization mechanisms can be specified and enforced across organization boundaries, which gives companies enough confidence to expose their internal business operations as web services and participate in web services orchestration.

Semantic Web is to provide a common framework for data sharing and reuse across applications, enterprises, and community boundaries. Being a key enabling technology of semantic web, ontology facilitates data sharing by defining the terms and their relationships for a particular domain (W3C 2004). Rule markup language has been a mainstream research topic in semantic web, resulting in several business rule markup languages, such as XRML (Lee et al. 2003) and RuleML (Grosof 2001). In particular, RuleML is based on situated courteous logic, which provides a strong theory foundation and some unique features such as prioritized conflict handling and disciplined procedural attachments (Grosof 2001). Moreover, tools have been developed to map RuleML to prolog and production rules, which enables RuleML to run on most well-known rule-based engines, such as XSB, Jess, and Jena (Grosof et al. 2003). The logic-based process modeling languages discussed in section 2.2.3 cannot be understood and exchanged among e-business partners. Given the features and analytical capability of RuleML, it is a good candidate XML-based representation language for process constraints. As we will discuss
in section 7, the techniques and standards reviewed in this section provide a set of enabling
technologies for the development of our prototype system.

2.5 Process Change and Adaptive Workflow

By automating well-defined business processes, workflow management systems have
delivered increased productivity, faster cycle time, less errors, and lower operation costs.
However, while workflow applications grow in number and size, organizations are
recognizing that in order to release the real power of process automation workflow
management systems must be able to adapt efficiently and effectively to changes in
business and the needs of the people (Kammer et al. 2000; Ultimus 2006). As such,
research in dynamic and adaptive workflow has received increasing attention in recent
years (Klein et al. 2000b).

Workflow changes can be made to workflow model or workflow instance (Sadiq et al.
2000). Workflow instances are particular occurrences of a workflow model. The changes to
workflow model are often permanent as the result of process improvement, process
redesign or reengineering, merger/acquisitions, etc., whereas changes in individual or a
few workflow instances are usually due to unforeseen and rare situations in process
operations (Sadiq et al. 2000). If not properly handled, workflow changes can cause
inefficiencies, inconsistencies, high runtime costs, or catastrophic enterprise system
breakdowns (Ellis et al. 1995).

When workflow changes happen, different actions can be taken to handle the changes,
which have different impacts to the existing workflow instances. Sadiq et al (2000) defined
five workflow modification policies to cope with workflow changes, namely, Flush, Abort, Migrate, Adapt, and Build. Specifically, in flush situations, all current workflow instances are permitted to complete according to the old process model, while new instances must follow the new model. Active workflow instances may also be aborted, which may incur losses to the organization without compensating for the tasks already accomplished when abortion is executed. Migrate method aims to introduce the changes to running instances without either aborting or flushing. This kind of migration involves thorough analysis of process structure and current process status in order to avoid process errors known as “dynamic change bugs” (van der Aalst 2001). In particular, several mathematical models have been proposed to formally represent workflow dynamic changes and identify “safe” ways to migrate existing instances without incurring dynamic change bugs (Ellis et al. 2000; Reichert et al. 1998; van der Aalst 2001). Furthermore, a set of correctness criteria for dynamic workflow changes has been summarized to help validate process change migration results (Rinderle et al. 2003).

Process change adaptation means that there are some workflow instances that need to be treated differently due to exceptions. Exceptions are defined as “deviations from an ideal collaborative workflow process caused by errors, failures, resource or requirements changes” (Klein et al. 2000a). Different methods for exception specification and handling have been proposed in the literature, such as knowledge-based approach (Klein et al. 2000a), active rule based approach (Casati et al. 1999), and meta modeling approach (Chiu et al. 1999). The importance of flexibility of workflow systems has also been recognized by industry vendors. For instance, one of the leading BPM software companies Ultimus
has incorporated a patented technology named Adaptive Discovery © to enable runtime deviation from pre-defined process model to cope with dynamic process changes (Ultimus 2006).

Although there has been noticeable amount of research in process change and adaptive workflow, most research only focus on change analysis of one process aspect, especially on control flow. Research on the change relationship among different process aspects such as control flow, data flow, organizational model and process constraints has been scant. In this dissertation, we investigate how changes in one process aspect can affect other aspects. In particular, we formally define various process constraint change anomalies due to changes in control flow and organizational model, and develop algorithms to automatically detect those anomalies as discussed later in Section 6.
3. A POLICY PERSPECTIVE ON PROCESS ANALYSIS AND DESIGN

Most organizations have a set of business policies that govern the way they conduct their business, e.g. twelve organization-wide policies are identified in (Peltier 2004), such as Employee Standards of Conduct, Workplace Security, Business Continuity Planning, etc. (Figure 3). Each organization-wide policy may also include many more specific sub-policies. Due to the trend towards e-business, organizations are also digitalizing their business policies and publishing those policies on the Internet for easy access and dissemination (Wang et al. 2005). For instance, Figure 4 shows an online business policy manual. Many of these business policies are used to define business processes often in terms of standard procedures, e.g., procedures for travel reimbursement, procedures for purchasing, procedures for payroll and cash handling, etc.

![Figure 3. An example of corporate policies (adapted from (Peltier 2004) page 4.)](image-url)
Although these process policies are the key elements for efficient running and control of organizations, many organizations do not have their business processes clearly documented, and in some cases their business processes are not documented at all. Lack of process documents may lead to inefficiency, misunderstanding across different functions and, more seriously, business fraud such as the financial scandals of WorldCom and Enron (Bloem et al. 2006). As such, recent government regulations related to Sarbanes-Oxley require organizations to clearly define and document their processes in organizational policies in order to increase internal control and governance (Cobb 2004). Furthermore, organizations are investing a great amount to revamp their business processes in order to be compliant with quality control standards such as ISO9001. As a result, many organizations have volumes of process policies in place.

Figure 4. Screenshot of an online business policy manual
3.1 The Policy Mismatch Problem in Business Process Management

As shown in Figure 5, high-level narrative business policies are usually not directly mapped into the field operations, resulting in a missing link between business policies and process models. In today’s dynamic business environment, organizations often need to modify their business processes to cope with ever-changing customer demand and market opportunities. In addition, organizations are investing a great amount to revamp their business policies to comply with recent regulations related to Sarbanes-Oxley. Because of the missing link between business policies and process models, policy changes cannot be efficiently propagated to the operational level. For the same reason, real process operations often deviate from defined policies after several process changes. We refer to this problem as the policy mismatch problem in business process management. As such, for organizations with large-scale business processes, maintaining compatibility between business policies and process models can be a very challenging task. Therefore, a systematic linkage between business policies and process models must
be establish to provide solutions.

The policy aspect of business process has been largely overlooked in the literature, and how to systematically retrieve process information from business policies to build process models has not been studied. Next, we present a case study of a major public university’s business policy manual and show how process information can be identified from those policies by applying a process policy taxonomy.

### 3.2 A Case Study of Business Policies

In this section, we present a case study of the business policy manual of a major American public university. This business policy manual is published online and is accessible to the public as shown in Figure 6.

**Figure 6. Screenshot of travel policy section in an online manual**

The policy manual has nineteen sections covering various organizational aspects, such as accounting, finance, information systems, etc. For the case study, we focus on
“Travel Regulations” section, because travel approval and reimbursement processes are implemented in most organizations and are therefore representative. Furthermore, these processes involve many different scenarios, rules, and constraints, which make the corresponding process model non-trivial.

As we discussed in the previous section, we are interested in how high-level process polices are linked to operational process models. In this study, we chose travel approval and reimbursement processes as the operational process model. To see how these processes are defined in the policies, we summarize some key sentences and paragraphs from the manual:

- Heads of academic and administrative units authorize travel for their faculty and staff.
- All claims for the reimbursement of expenses for approved university business travel are made on a Travel Reimbursement Form.
- There is a list of reimbursable and non-reimbursable expenses.
- There is detailed information on lodging and meal allowance.
- The Travel Reimbursement Form must be submitted promptly, preferably within 30 calendar days, but no later than 60 days, after completing the travel.
- If the Travel Reimbursement Form is not submitted within 60 days of the completion of travel, the reimbursement is considered taxable income to the traveler, unless a reasonable justification for an exception is presented.
- If a reasonable justification for an exception is not submitted or not approved, the reimbursement amount will be submitted to University Payroll and included on the employee's Form W-2.
• If the traveler submits a Travel Reimbursement Form for reimbursement more than 60 days after the completion of travel but feels he/she has a reasonable justification for the late submission, the Reasonable Exception Request must be submitted as well.

• The completed and approved Travel Expense Reimbursement Form should be mailed to the Disbursement Services Center.

• After the Travel Expense Reimbursement Form is reviewed for compliance with University/State policies, a check is issued to the traveler in less than 10 working days.

• The Employee must submit a Travel Exception Request Form if the reimbursement request is not compliant with university policies, e.g. total amount exceeds the reimbursement limit.

• Exceptions are not considered without the approval of the department head or delegate.

• The Office of Business and Financial Services reviews and records all exceptions claimed on travel vouchers.

• All exceptions are reported quarterly to the Higher Education Travel Control Board for review. If the exception is denied, the traveler must refund the disallowed amount to the university account charged when the employee was reimbursed.

Given these narratives policies, we ask the following questions:

• What process information can be identified from these policies?

• What are the types of process policies?

• Can these narrative policies be formalized and systematically analyzed?

• What is the procedure to transform process policies into workflow models?
- What changes can happen to process policies?

- How can the workflow model be affected by process policy changes?

The rest of this dissertation will answer these questions. Next, we propose a taxonomy of process policies to help identify process information from business policies.

3.3 Process Policy Taxonomy

Based on the study of several business policy manuals, we discovered that rich information for workflow models can be found in business policies. Given that a workflow model consists of four major components, namely, control flow, data flow, organizational model, and constraints, we classify process policies into four categories corresponding to those components (Basu et al. 2002). Each category is further divided into sub-categories to provide more detailed characteristics on different process policies to facilitate process policy identification. The resulting process policy taxonomy is shown in Figure 7.

![Figure 7. Process policy taxonomy](image-url)
Control flow policy has three types: task identification policy, task dependency policy, and routing constraint identification policy. For instance, a policy saying “all claims for the reimbursement of travel expenses are made on a Travel Reimbursement Form” is a task identification policy which specifies a task, “Travel Reimbursement Form submission.” Task dependency policy can be used to determine the task execution sequence. For example, a policy stating that “after the Reimbursement Form is approved, a check will be issued” identifies that the task “issue check” should be executed after the task “Reimbursement Form approval”. A policy can be used to identify routing constructs, such as parallel execution or decision points, is a routing construct identification policy. For example, a policy stating that “if reimbursement amount exceeds the limit, a Travel Exception Form must be filled out” specifies a routing condition: the next task is determined by whether or not the reimbursement amount exceeds the limit. Data flow and resource policies are defined similarly.

Process constraints restrict the execution of processes. In particular, three types are defined: assignment policies, routing constraints, and process rules. Assignment policies specify how each resource is assigned to each task and how different routing constraints are associated with routing constructs. For instance, a sample assignment policy can be “department head must approve the travel” which assigns a role “department head” to execute a task “travel approval.” Routing constraints are defined to decide the execution path when a decision point is met. Normally, policies with an “if then” format indicate routing conditions. Process rule policies define additional constraints that the process execution must follow. Examples of process rules include well-know dynamic
authorization constraints, such as separation or binding of duties, temporal constraints, such as “task t must be completed by 5pm on July 25th, 2006”, and exception handling constraints, such as “if the dean will not be able to sign the form within two weeks, the vice dean is authorized to sign on his behalf.”

By applying the process policy taxonomy to the case presented in the previous section, we identify the following control flow policies:

- **P1**: All claims for reimbursement of expenses for approved university business travel are made on a Travel Reimbursement Form.
- **P2**: If the Travel Reimbursement Form is submitted more than 60 days after completing the travel, a Reasonable Exception Request Form must be submitted.
- **P3**: If the Reasonable Exception Request is not approved, then the reimbursement amount will be taxable, showing on employee’s Form W-2.
- **P4**: If the reimbursement amount exceeds the limit, a Travel Exception Form must be filled out.
- **P5**: After the reimbursement form is approved, a check will be issued.
- **P6**: If the quarterly travel exception review is denied, the reimbursed amount must be refunded.

From these policies, the following tasks can be identified:  
- **t1** submit Travel Reimbursement Form,  
- **t2** approve travel,  
- **t3** submit Reasonable Exception Form,  
- **t4** review reasonable exception,  
- **t5** make reimbursement taxable,  
- **t6** submit Travel Exception Form,  
- **t7** review Travel Exception Form,  
- **t8** review Travel Reimbursement Form,  
- **t9** issue check,  
- **t10** quarterly review travel exception,  
- **t11** refund back the reimbursement check. In addition, the
following routing constructs (decision points/branches) are identified: b₁ whether or not the reimbursement form is submitted more than 60 days after completion of travel, b₂ whether or not reasonable exception request is approved, b₃ whether or not the reimbursement amount exceeds the limit and b₄ whether or not the quarterly travel exception review is denied. Furthermore, some sequential execution orders between tasks and decision points are also indicated in the policies. For example, P2 indicates that t₃ is executed after b₁. Based on the control flow policies, a preliminary control flow diagram of the travel reimbursement process can be developed as illustrated in Figure 8.

We can also extract two data flow policies from the case study as listed below:

- **P7:** There is a list of reimbursable and non-reimbursable expenses (e.g., alcoholic beverages are a non-reimbursable expense).
- **P8:** There are detailed lodging, meal, and per diem allowances (e.g., the lodging allowance in the Chicago metropolitan area is $149).

![Figure 8. Preliminary travel reimbursement process based on control flow policies](image-url)
Note that P7 and P8 are actually two sets of similar data flow policies. These data will be used to review the Reimbursement Form. Some other data items are implicitly identified in the control flow policies, including Travel Expense Reimbursement Form, Reasonable Exception Form, Travel Exception Form, and Check. This also demonstrates that different types of business process policies are not exclusive and can be overlapping in terms of their functionalities.

The following shows the role-task assignment policies defined in the case policies:

- P9: Department or unit head must approve the travel.
- P10: Department or unit head must approve the Reasonable Exception Request.
- P11: Disbursement Services Center (DSC) must review the Reimbursement Form.
- P12: The Office of Business and Financial Services (OBFS) reviews the Travel Exception Form.
- P13: The Higher Education Travel Control Board (HETC) reviews the travel exceptions quarterly.

Four organization resources are identified: department head, Disbursement Services Center, Office of Business and Financial Services, and Higher Education Travel Control Board, each of which is assigned to a particular task. In addition, a process rule is also identified, which is a temporal constraint - P14: It is preferred that the Reimbursement Form is submitted with 30 days upon completing the travel. Note that P14 is a “soft” performance policy, which is not always enforced. In contrast, the violation of some mandatory process rules may cause process execution exceptions.

In summary, the process policy taxonomy presented in this section classifies process
policies into control flow policies, data flow policies, resource policies, and constraint policies. This taxonomy is sufficiently inclusive to contain most commonly used process policies and is also concise and extendible. By applying this taxonomy to the case study, we identified 11 tasks, numerous data items, and 14 process policies, which help derive a preliminary process model as shown in Figure 8.

This case study demonstrates that rich process information can be extracted from narrative business policies to help construct process models. However, those process policies are still not precisely represented and cannot be formally analyzed. The process model extraction procedure is not well defined, and the criteria for a process model to be complete have not been discussed. In the next section, we go one step further in this research direction by formalizing the process extraction procedure.
4. POLICY-DRIVEN PROCESS DESIGN

In this section, we propose an innovative process design methodology named Policy-driven Process Design (PPD). PPD takes narrative business policies as process design input and generates process models by formalizing and parsing those policies. One unique feature provided by PPD is the formal linkage between business policies and workflow models, which enables efficient and systematic management of policy-process consistency under various changes. Furthermore, PPD also builds the foundation for policy-based process design automation.

Next, we first present overall PPD procedure, followed by the formalization of process policy. Then, we discuss the process model extraction process, where a set of new concepts are introduced, such as task view, data view, and process design rules. A case study is conducted to validate our policy-driven process design methodology and the architecture of PPD prototype system is also discussed.

4.1 Policy-driven Process Design Procedure

As shown in Figure 9, PPD consists of four major steps, which are defined as follows:

- Extract process policies. Most organizations have a standard set of policies that govern the way they conduct their business, which we refer to as business policies (Peltier 2004). Among those business policies, some policies are not related to business processes such as workplace security policies, employee standards of
conduct, etc. The first step for process analysts using PPD approach is to distill process policies from general business policies and express them in a succinct and clear format. This step greatly relies on human intelligence and related training may be needed for better process policy extraction results.

![Figure 9. Policy-driven process design procedure](image)

- Applying process policy template. The process policies identified in step one are narrative sentences. In order to facilitate policy analysis, those process policies need to be formalized using process policy template. In particular, according to the process policy template, process policies are precisely decomposed into information on task, task dependency, task execution order, data items, data dependency, and process constraints. This step is also conducted by process analysis with the assistant of a process parsing tool and all formalized process policies information is stored in
database for further analysis.

- Execute process extraction procedure. In this step, a set of process design algorithms and rules are applied to the formalized process policies to construct a preliminary process model. For example, identified tasks are first linked based on the execution ordering information extracted from the policies, and then additional links among tasks are added based on the data dependency analysis. Part of this step is automated via a set of algorithms, such as initial control flow generation, task dependency analysis, data flow analysis, process completeness checking, etc. However, revising process model according to design rules and removing potential semantic errors still need human intelligence. A visualization tool is developed to help process analysts better understand and enhance the process model.

- Refine process model. In this step, the preliminary process model is reviewed and refined for completeness and correctness. Additional process data collections may be needed if the process policies cannot provide all necessary information to finalize the process model.

Given that step one and four are mainly based on human intelligence, we focus our discussion on step two and three, where computer technology can help improve process design efficiency and accuracy. Next, we first present the formalization of process policy and process map.

4.2 Process Policy Template and Process Map Concepts

As we discussed in section 3, process policies consists of information on control flow,
data flow, organizational model, and process constraints. Although those narrative process policies are intuitive and easy for human to understand, they cannot be interpreted by computers until they are formalized into a precise and rigorous format. Similarly, process models also need to be formally defined and represented. In this section, we use process policy template and process map to formalize process policies and process models respectively. First, process policy template is formally defined as follows:

**Definition 1 (Process Policy Template).** Let \( P \) be a finite set of process policies, \( P = \{p_1, p_2, \ldots, p_n\} \). We define a process policy template \( p_i \in P \) as an 8-tuple: \( p_i = <\text{pid}_i, D_i, T_i, C_i, R_i, TSI_i, TR_i, TDi_i> \), where

- \( \text{pid}_i \) is the process policy ID, which is unique in order to identify a policy:
  \[
  \forall p_i, p_j \in P, i \neq j, \text{pid}_i \neq \text{pid}_j
  \]
- \( D_i = \{d_1^i, d_2^i, \ldots, d_m^i\} \) is the set of data items related to \( p_i \). \( D = D_1 \cap D_2 \cap \ldots \cap D_n \) represents all data items in all process policies.
- \( T_i = \{t_1^i, t_2^i, \ldots, t_l^i\} \) is the set of tasks related to \( p_i \). \( T = T_1 \cap T_2 \cap \ldots \cap T_n \) is the set of all tasks included in all process policies.
- \( C_i = \{c_1^i, c_2^i, \ldots, c_k^i\} \) is the set of constraints related to \( p_i \). \( C = C_1 \cap C_2 \cap \ldots \cap C_n \) is the set of all constraints. \( c_k^i \) is either an expression or a statement that returns a Boolean value.
- \( R_i = \{r_1^i, r_2^i, \ldots, r_s^i\} \) is the set of resources related to \( p_i \). \( R = R_1 \cap R_2 \cap \ldots \cap R_n \) is the set of all resources. \( r_s^i \in U \cap RL \cap OC \cap AG \), where \( U \) is set of users, \( RL \) is set of
roles, e.g., general manager, salesperson, accountant, etc., OC is set of organizational units, e.g., bursar’s office, graduate college, etc., and AG is set of machine agents, e.g., accounting system, ERP, etc.

- \( TS_i = T_i \times T_i \times C_i = \{(t_a^i, t_b^i, c_e^i) : t_a^i \text{ is executed before } t_b^i \text{ if } c_e^i \text{ is true}\} \) is the set of task sequences related to \( p_i \).
- \( TR_i = T_i \times R_i = \{(t_i^i, r_k^i) : t_i^i \text{ is executed by } r_k^i\} \) is the set of task resource relationships in \( p_i \).
- \( TD_i = T_i \times D_i = \{(t_i^i, d_m^i) : d_m^i \text{ is used in } t_i^i\} \) is the set of task data relationships in \( p_i \).

The goal of process policy template is to parse narrative process policies into small pieces of precise information on different workflow components. For instance, a travel reimbursement policy: “If Travel Reimbursement Form is submitted later than 60 days upon completing the travel, a Reasonable Exception Request Form must be submitted” can be formalized according to process policy template as follows. Assume the \( pid \) for this policy is 1. Then, two data items can be identified: \( d_{11} = \text{“Travel Reimbursement Form (TRF)”} \) and \( d_{21} = \text{“Reasonable Exception Request Form (RERF)”} \). Two tasks can be extracted: \( t_{11} = \text{“Submit TRF”} \) and \( t_{21} = \text{“Submit RERF”} \). There is one constraint: \( c_{11} = \text{“TRF submission later than 60 days upon completing the travel”} \). In summary, this policy can be expressed as \( p_1 = \langle 1, \{d_{11}, d_{21}\}, \{t_{11}, t_{21}\}, \{c_{11}\}, \emptyset, \{(t_{11}, t_{21}, c_{11})\}, \emptyset, \{( t_{11}, d_{11}),( t_{21}, d_{21})\} >. \) Because no resources can be identified by this policy, the corresponding sets are empty. As we will discuss later in the system architecture and prototyping chapter, tools are developed to help process analyst parse process policies according to the template. In particular, the system automatically indexes and records identified workflow
elements to avoid duplications and support efficient search.

The result of process design is a process model. To simplify and facilitate policy-driven process design, we introduce the concept of process map, which is a process model defined as a directed graph with some special properties as discussed below (Wilson 1996).

**Definition 2 (Process Map).** Given process policy template \( p_i = \langle \text{pid}_i, D_i, T_i, C_i, R_i, TS_i, TR_i, TD_i \rangle \), a process map is defined as a directed graph \( G = \langle V_t(G), V_c(G), V_{se}(G), A(G) \rangle \), where:

- **task vertex set** \( V_t(G) = \{ (v_t, D_t, R_t) : v_t \in T, D_t \subset D, R_t \subset R, D_t \neq \emptyset, R_t \neq \emptyset \} \) is a non-empty finite set of all tasks in the process map and each task is associated with a set of data items \( D_t \) and resources \( R_t \). \( d^-(v_t), d^+(v_t) \geq 1, d^-(v_t) = 1 \), where \( d^-(v_t) / d^+(v_t) \) is \( v_t \)'s in-degree/out-degree.

- **decision vertex set** \( V_c(G) = C \) is a finite set of all constraints in the process map, and \( \forall v_c \in V_c(G), d^-(v_c) \geq 1, d^+(v_c) = 2 \).

- **start/end vertex set** \( V_{se}(G) = \{ s, e \} \), where \( s \) is the start node and \( e \) is the end node in the process map, and \( d^-(s) = 0, d^+(s) = 1, d^-(e) \geq 1, d^+(s) = 0 \).

- **directed arc set** \( A(G) \) is a non-empty finite set of distinct ordered pairs of distinct elements of vertex set \( V(G) = V_t(G) \cup V_c(G) \cup V_{se}(G) \).

Besides enabling formal process analysis, Definition 2 also implies some assumptions we make in order to simplify the structure of process maps: 1) tasks are executed sequentially, and task parallelism can be introduced later to improve process
performance; 2) each decision vertex has two and only two outgoing arcs, which means that each decision vertex is an exclusive OR and its value is a Boolean value returned by corresponding process constraint. It has been proved that any OR vertex can be converted to exclusive OR vertex (Bi et al. 2004a); 3) a task can have only one outgoing arc and at least one incoming arc; 4) an end node can have more than one incoming arcs. As we will discuss later, a set of rules are developed to ensure that the process map follows the assumptions. Next, we show a systematic procedure to extract process maps from process policies.

4.3 Process Model Extraction Procedure
Figure 10 shows the procedure to extract a process map from formalized process policies. We propose a set of new concepts to facilitate the extraction procedure, namely, task view, data view, and process design rules. Each step of the procedure is discussed in detail as follows:

**Step 1: Draft process map based on task view.** In this step, the first draft of process map is generated based on the task sequences $TS$ identified from policy analysis. In particular, a task view is first constructed based on $TS$ to simplify the drafting process.

<table>
<thead>
<tr>
<th>Algorithm 1:</th>
</tr>
</thead>
</table>
| **Input:** initial process map $G = (V_i(G), V_c(G), V_{sc}(G), A(G))$<br>where, $V_i(G) = V_c(G) = V_{sc}(G) = A(G) = \emptyset$,<br>and task view $TV$, where $T = \{t_1, t_2, ..., t_n\}$, $C = \{c_1, c_2, ..., c_m\}$<br>**Output:** draft process map $G'$<br>**Begin**<br>$V_{sc}(G) = \{s, e\}$ // add start and end node<br>$V_c(G) = C$ // add decision points<br>$V_i(G) = \{(v_i, D, R_i) : v_i \in T, D = R_i = \emptyset\}$ // add tasks<br>for each $tv_{ij} \in TV$ // add arcs among tasks and decision points based on task view<br>if $tv_{ij} = 1$ and $i < n, j < n$ then add $(t_i, t_j)$ to $A(G)$<br>if $tv_{ij} = 1$ and $n < i < n + m, j < n$ then add $(c_i, t_j)$ to $A(G)$<br>if $tv_{ij} = 1$ and $i < n, n < j < n + m$ then add $(t_i, c_j)$ to $A(G)$<br>if $tv_{ij} = 1$ and $n < i < n + m, n < j < n + m$ then add $(c_i, c_j)$ to $A(G)$<br>**end for**<br>$G' = (V_i(G), V_c(G), V_{sc}(G), A(G))$<br>**End**

Table 1. Algorithm for drafting process map based on task view

**Definition 3 (Task View).** Given $T = \{t_1, t_2, ..., t_n\}$ are identified tasks and $C = \{c_1, c_2, ..., c_m\}$ are identified constraints. A task view $TV$ is defined as a $(m+n+1)$-square matrix, whose rows and columns correspond to the set $T \cup C \cup \{s\}$ and
$T \cup C \cup \{e\}$ where $s$ is the start node and $e$ is the end node. $tv_{ij} = 1$ and if the corresponding task, constraints, or start/end nodes are connected, otherwise $tv_{ij} = 0$.

Given the initial process map $G$, which is an empty graph, and task view $TV$, the following algorithm is used to draft the process map.

Based on the task view, a preliminary process map can be easily drafted. For instance, a $TV$ matrix with three tasks and two constraints is mapped into a draft process map in Figure 11. Normally, task view only provides limited information on process map, and additional enhancements and refinements are needed to complete the process map.

<table>
<thead>
<tr>
<th>TV</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>c₁</th>
<th>c₂</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₂</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c₁</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c₂</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 11. Task view example

**Step 2: Refine the Process Map based on Data View.** In this step the draft process map $G'$ is enhanced based on the identified task data relationship $TD$. A data view is constructed to help this process, which is defined as following:

**Definition 4 (Data View).** Given $T = \{t₁, t₂,..., tₙ\}$ are identified tasks, $C = \{c₁, c₂,..., cₘ\}$ are identified constraints, and $D = \{d₁, d₂,..., dₖ\}$ are identified data items. A data view $DV$ is defined as a $(m+n) \times k$ matrix whose rows correspond to the set $T \cup C$, and whose columns correspond to the set $D$. $dv_{ij} = 0$ ($dv_{ij} = 1$) if $d_j$ is the output
(input) of corresponding task or constraint, otherwise $d_{ij} = 0$.

Based on data view, additional arcs between tasks and decisions can be identified. Intuitively, if a data item $d$ is the output of task $t_1$ and input of $t_2$, then an arc $(t_1, t_2)$ should be added. Formally, the following algorithm is used to refine process map $G'$. Note that the task view is rebuilt based on data view at the end of the algorithm.

**Algorithm 2:**

Input: draft process map $G = \langle V_t(G), V_c(G), V_{sc}(G), A(G) \rangle$

data view $DV$, where $T = \{t_1, t_2, \ldots, t_n\}$, $C = \{c_1, c_2, \ldots, c_m\}$, $D = \{d_1, d_2, \ldots, d_k\}$

Output: refined process map $G$

Begin

var TAIL // a set variable to store tails for new arcs
var HEAD // a set variable to store heads for new arcs

for $j = 1 \ldots k$

for $i = 1 \ldots (n + m)$

$dv_{ij} \in DV$

if $dv_{ij} = O$ and $i < n$ then add $t_i$ to TAIL

if $dv_{ij} = O$ and $n < i < n + m$ then add $c_i$ to TAIL

if $dv_{ij} = I$ and $i < n$ then add $t_i$ to HEAD

if $dv_{ij} = I$ and $n < i < n + m$ then add $c_i$ to HEAD

end for

end for

for $tail \in TAIL, head \in HEAD$

if $(tail, head) \notin A(G)$ then add $(tail, head)$ to $A(G)$

end for

$G' = \langle V_t(G), V_c(G), V_{sc}(G), A(G) \rangle$

run Algorithm 1 // rebuild the task view based on data view

End

**Table 2. Algorithm for process map enhancement based on data view**

For example, Figure 12 shows how the draft process map in Figure 11 can be enhanced based on a data view.
Step 3: Refine process map based on Process Map Rules (PMRules). The previous two steps try to get as much information on the process map as possible by adding arcs among tasks and decision points, which may result in a process map that does not confine to the process map definition specified in Definition 2. Such process maps are usually not well organized and hard to understand. Thus, we say a process map is syntactically correct, if it satisfies all process map structure properties defined in Definition 2, which is formally specified in Definition 5.

Definition 5 (Syntactical Correctness of Process Map). Given task view $TV$, where $T = \{t_1, t_2, \ldots, t_n\}$, $C = \{c_1, c_2, \ldots, c_m\}$, a process map is syntactically correct if and only if:

$$i = 1, \ldots, n, n + m + 1 \sum_{j=1}^{n+m+1} tv_{ij} = 1 \text{ // single outing arc for tasks and start node}$$

and $j = 1, \ldots, n + m + 1, \sum_{i=1}^{n+m+1} tv_{ij} \geq 1 \text{ // at least one incoming arc for tasks, decisions, and end node}$

and $i = n + 1, \ldots, m, \sum_{j=1}^{n+m+1} tv_{ij} = 2 \text{ // two outgoing arcs for decisions}$
In this step, the enhanced process map $G''$ is further refined according to a set of PMRules for syntactical correctness. In particular, the following PMRules are enforced:

**PMRule 1 (Single Task Outgoing Arc).** If there exists tasks that have more than one outgoing arcs, those links must be merged. Basically, this rule ensures the proper execution of a process by avoiding ambiguous execution paths. Given tasks in $G''$ are connected to either other tasks or decisions, we use the following two sub PMRules to enforce PMRule 1.

**PMRule 1.1 (No One-to-Many Task-Task Arcs).** If a task $t$ links to more than one other tasks directly, the other tasks must be sequentially linked and connected to $t$.

**PMRule 1.2 (No One-to-Many Task-Decision Arcs).** Similar to PMRule 1.1, if a task $t$ links to more than one decision points directly, those decision points must be sequentially linked and connected to $t$.

**PMRule 2 (Boolean Decision Outgoing Arc).** If a decision point has less than two outgoing arcs, additional arcs need to be added to link the decision point to other tasks or decision points. Similarly, if there are more than two outgoing arcs, the decision point should be split into multiple ones. According to process policy definition, a constraint returns a Boolean value and each decision point is associated with one constraint. Therefore, a decision point must have two and only two outgoing arcs.

**PMRule 3 (Close Start/End Node).** All tasks and decisions without incoming arc should be connected to the start node; and all tasks without outgoing arc should be connected to the end node; After that, if more than one task/decision is linked to the start node, those tasks/decisions need to be sequentially linked first and then connected to the start node;
and if there are still no arcs linked to start and end nodes, link them directly.

**Step 4: Remove semantic errors.** Semantic error is defined as inappropriate arcs resulting in process maps that are logically wrong. For example, a direct arc from start node to end node represents a semantic error. In this step, a thorough walkthrough of the process map should be conducted to identify and remove semantic errors if any. Semantic errors inspection relies mainly on human intelligence, although some simple errors, such as the start-to-end arc aforementioned, can be identified by computers. We defer the thorough investigation of process design semantic errors to future research.

**Step 5: Process review.** In this step, the process map is reviewed to ensure it captures all tasks and constraints required by corresponding policies. Process review usually includes review meetings with process owners, e.g., managers who oversee the process, and people who are conducting process operations. Given that a process map has been developed based on existing process policies, this review is more focused and efficient than data collections in traditional participative approach. It is also possible that the business policies are out of date. In that case, the extracted process map is revised to match the real process in the field and corresponding business policies are also updated.

To the best of our knowledge, the policy-driven process design procedure presented in this section is the first attempt to formalize the procedure of process design based on policy analysis. PPD advocates a new process mapping methodology and creates a critical step towards process design automation. In the next section, we demonstrate and validate the PPD approach via a case study.
4.4 Validation of Policy-driven Process Design

In section 3, we conducted a case study of a major public university’s business policy manual, from which we identified a set of process policies as shown in Table 3 (Wang et al. 2005). In this section, we show how those twelve narrative process policies are transformed into a process map using the PPD approach.

| P1: All claims for the reimbursement of expenses for an approved university business travel are made on a Travel Reimbursement Form (TRF). |
| P2: If Travel Reimbursement Form is submitted later than 60 days upon completing the travel, a Reasonable Exception Request Form (RERF) must be submitted. |
| P3: If reasonable exception request is not approved, the reimbursement amount will be taxable. |
| P4: If reimbursement amount (RA) exceeds limit, a Travel Exception Form (TEF) must be filled. |
| P5: After the reimbursement form is approved, a check will be issued. |
| P6: If the quarterly travel exception review is denied, the reimbursed amount must be refunded. |
| P7: Department or unit head must approve the travel. |
| P8: Department or unit head must approve the reasonable exception. |
| P9: Disbursement Services Center (DSC) must review the reimbursement form. |
| P10: The Office of Business and Financial Services (OBFS) reviews the travel exception form. |
| P11: Higher Education Travel Control Board (HETC) quarterly reviews the travel exceptions. |
| P12: Bursar’s office (BO) is responsible to issue reimbursement check. |

Table 3. Identified process policies from a business policy manual

First, we formalize the policies according to process policy definition specified in
Definition 1 as shown in Table 4. In particular, eleven tasks, five constraints, five data items, and six resources are identified.

<table>
<thead>
<tr>
<th>PID</th>
<th>Data Items (D)</th>
<th>Tasks (T)</th>
<th>Constraints (C)</th>
<th>Resource (R)</th>
<th>Task Sequences (TS)</th>
<th>Task Resources (TR)</th>
<th>Task Data (TD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d1 = TRF</td>
<td>t1 = Submit d1</td>
<td>NA</td>
<td>r1 = traveler</td>
<td>(t1, r1)</td>
<td>(t1, d1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d2 = RERF</td>
<td>t1 = Submit d2</td>
<td>c1: TRF is submitted later than 60 days upon completing the travel</td>
<td>NA</td>
<td>(t1, t2, c1)</td>
<td>(t2, r1)</td>
<td>(t2, d2)</td>
</tr>
<tr>
<td>2</td>
<td>d2 = RA</td>
<td>t3 = Approve d2, t4 = Make d3 taxable</td>
<td>c2: RERF is approved</td>
<td>NA</td>
<td>(t3, t4, c2)</td>
<td>NA</td>
<td>(t3, d2)</td>
</tr>
<tr>
<td></td>
<td>d3 = TEF</td>
<td>t5 = Submit d4</td>
<td>c3: Reimbursement Amount exceeds the limit</td>
<td>NA</td>
<td>(null, t5, c3)</td>
<td>(t5, r1)</td>
<td>(t5, d4)</td>
</tr>
<tr>
<td>3</td>
<td>d1 = Check</td>
<td>t6 = Approve d1, t7 = Issue d5</td>
<td>c4: TRF is approved</td>
<td>NA</td>
<td>(t6, t7, c4)</td>
<td>NA</td>
<td>(t6, d1) (t7, d6)</td>
</tr>
<tr>
<td></td>
<td>d4, d5</td>
<td>t8 = Review d4 quarterly, t9 = Refund d5</td>
<td>c5: TEF quarterly review is not passed</td>
<td>NA</td>
<td>(t8, t9, c5)</td>
<td>(t9, r1)</td>
<td>(t8, d4) (t9, d5)</td>
</tr>
<tr>
<td>4</td>
<td>d1</td>
<td>t6</td>
<td>NA</td>
<td>r2 = Dept. Head</td>
<td>NA</td>
<td>(t6, r2)</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>d2</td>
<td>t3</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>(t3, r2)</td>
<td>NA</td>
</tr>
<tr>
<td>6</td>
<td>d1</td>
<td>t10 = Review d1</td>
<td>NA</td>
<td>r3 = DSC</td>
<td>NA</td>
<td>(t10, r3)</td>
<td>NA</td>
</tr>
<tr>
<td>7</td>
<td>d4</td>
<td>t11 = Review d4</td>
<td>NA</td>
<td>r4 = OBFS</td>
<td>NA</td>
<td>(t11, r4)</td>
<td>NA</td>
</tr>
<tr>
<td>8</td>
<td>d4</td>
<td>t8</td>
<td>NA</td>
<td>r5 = HETC</td>
<td>NA</td>
<td>(t12, r5)</td>
<td>NA</td>
</tr>
<tr>
<td>9</td>
<td>d5</td>
<td>t7</td>
<td>NA</td>
<td>r6 = BO</td>
<td>NA</td>
<td>(t7, r6)</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 4. Formalized travel reimbursement process policies

Based on the task sequences in Table 4, a draft process map is constructed as shown in
Figure 13. In traditional participative process design projects, figuring out these tasks and task relationship may involve many rounds of interviews and meetings that can easily span over several days. With systematic parsing process policies, this draft process map can be accomplished by an individual process analyst in hours. It is worth noting that once Table 4 is generated, this step can be fully automated by visualization software, which greatly improves process design efficiency.

Figure 13. Draft process map based on the task sequences

However, the draft process map Figure 13 contains only limited information on the relationship among tasks and constraints, which is not enough for thorough understanding of the whole process. Therefore, we use data flow analysis to enhance the draft process map. In particular, Table 5 shows the data view of travel reimbursement process. Based on the algorithms in Table 2, additional arcs are added to represent more relationship among
tasks, and the enhanced process map is depicted in Figure 14. Compared with Figure 13, Figure 14 contains significant more information on the process, which is also automatically added by the process design tool.

<table>
<thead>
<tr>
<th></th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
<th>d5</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td></td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t6</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td></td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t8</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t9</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t10</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t11</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c4</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c5</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Data view of travel reimbursement process

However, according to process map definition, the enhanced process map is not syntactically correct. For instance, \( t_1 \) has five outgoing arcs and start node has no outgoing arc. Additional refinements are conducted using the PMRules. For example, task \( t_1 \) has no incoming arc, which should link to the start node. Similarly, \( t_9 \) has no outgoing arc, which should be linked to the end node. One outgoing arc is added to each decision point to represent the Boolean value returned by each constraint.

Compared with the previous two steps, this step needs human intervention and cannot yet be automated. However, some implicitly stated rules can help expedite the refinement process. For instance, one example implicit rule in our case study can be that before the travel reimbursement is reviewed, all kinds of exception forms must be submitted if any.
Using this rule, we can easily decide that task $t_{10}$ (review TRF) should be executed after all tasks related to TEF (travel exception form) and RERF (reasonable exception request form). Finally, the revised process map shown in Figure 15, which is syntactically correct. Note that in this step, some arcs identified in the task view and data view have been removed.

![Enhanced process map based on data view](image)

Figure 14. Enhanced process map based on data view

According to policy-driven process model extraction procedure, a syntactically correct process model may contain semantic errors. In our case, two semantic errors are identified in Figure 15: $t_7$ (issue the check) is linked to $t_9$ (refund back the check), which is logically wrong, and $t_{11}$ (Review TEF) is linked to $t_8$ (Review TEF quarterly), which results in the corresponding reimbursement request form never gets reviewed in $t_{10}$. To remove those semantic errors, two constraints $c_6$ and $c_7$ are added as shown in Figure 16.
A review meeting involving process analysts, process owners, and process users should be conducted to verify the process map in Figure 16. The goal is to make sure the process map reflects the real processes running in the field and it contains all the tasks and constraints. In addition, process map in Figure 16 only graphically illustrate the control flow aspect of a process. As shown in Table 4, all other information on data flow, resources, resource-task assignments should be provided as supplementary documents to the process map. In our case, this updated process map contains all the necessary tasks and constraints for the travel reimbursement process, leading to the final process map.

Figure 15. Revised process map based on PMRules

The process extraction procedure can also be depicted conveniently with a process matrix in Table 6. Initially, the process matrix contains the P’s based on the given process
policies. Then, the matrix is updated by adding D’s for the arcs identified from the data view in Table 5. Next, the semantic analysis removes a number of arcs as represented in the process matrix by the symbols of $P \rightarrow 0$ and $D \rightarrow 0$, respectively. The arcs added after applying all PMRules are depicted with the R’s. While the process maps in Figure 13 through Figure 16 are easy to read by process analysts, the process matrix is easier to automate with computer procedures. Note that the process matrix is closely related to the task view defined previously.

![Process map after removing semantic errors](image)

**Figure 16. Process map after removing semantic errors**

We have demonstrated the feasibility of Policy-driven Process Design (PPD) by
successfully extract a travel reimbursement process from the corresponding business policies. Besides being an efficient process design methodology, PPD is also of great benefit to organizations by improving the quality of their business policies. The quality of business policies can be measured in terms of preciseness, consistency, and completeness (Agarwal et al. 1993b; Teichroew et al. 1977). Business policies are written in natural languages that are not precise enough to describe process models and different readers may interpret same sentences in different ways. PPD greatly reduces the ambiguity of narrative process policies by precisely representing them using process policy template. For organizations with large-scale business processes, the corresponding process policies can be very large, which makes it very difficult to ensure the consistency of those policies. In PPD, once process policies are formalized, various analyses can be conducted to check policy consistency. For instance, a policy defining t₂ is executed after t₁ conflicts with another policy saying t₁ is executed after t₂. These types of conflicting policies can be easily identified using process matrix shown in Table 6 by finding non-zero values for matrix elements with index ij and ji.

A set of process policies are said to be complete if they provide all information needed to build the target process model. For large and complex business processes, it is difficult to determine what process information is missing by examining narrative process policies. Although completeness can never be fully guaranteed, PPD can facilitate detecting gaps and omissions by making them more obvious. For example, we defined syntactical correctness of process map in Definition 5 and one syntactical error is that there are tasks without outgoing arc, which means additional process information is
needed to determine how process will be routed after those tasks. Algorithms can be easily developed to check the syntactical correctness of a process map and therefore identify the missing process information.

In summary, we have shown that PPD can expedite process design by leveraging process policy analysis and improve the quality of business policies by formalizing narrative process policies. Next, we present the architecture of a prototype process design system developed based on Policy-driven Process Design methodology.

<table>
<thead>
<tr>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
<th>t10</th>
<th>t11</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>c4</th>
<th>c5</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6. Process matrix for travel reimbursement process design

4.5 Architecture of PPD Prototype System

Figure 17 shows the architecture of our web-based PPD prototype system, which consists of three major components, namely, Policy Wizard, Visual Editor, and Process Exchanger.
Policy wizard is a process policy analysis and parsing tool that provides step by step instructions for process policy formalization and extraction. For example, when a new process policy is entered, policy wizard provides intuitive interface for entering new identified tasks, task sequence information, data items, and task-data relationships. The system also keeps records of entered tasks, data items, process constraints and
automatically indexes those information to avoid duplicate data. Policy wizard is implemented as a set of JSP pages, and Figure 18 is a screenshot.

The formalized process policies are stored at a policy based, which is implemented using Oracle 10g ©. Figure 19 is the entity-relationship diagram of process policies, which is designed based on formal process policy definition presented in Definition 1 in section 4. Note that the cardinality between any two entities is many-to-many, which is not depicted in the figure. Then, the ERD is translated into a relation schema as shown in Table 7 with primary keys underlined.

<table>
<thead>
<tr>
<th>Policy (PID, PType, Keywords, Memo)</th>
<th>PolicyData(PID, DID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task (TID, TName)</td>
<td>PolicyResource(PID, RID)</td>
</tr>
<tr>
<td>DataItem (DID, DName)</td>
<td>PolicyConst(PID, CID)</td>
</tr>
<tr>
<td>PConstraint (CID, Content)</td>
<td>TaskSequence (TID1, TID2, CID)</td>
</tr>
<tr>
<td>PResource (RID, RName, Type)</td>
<td>TaskData (TID, DID)</td>
</tr>
<tr>
<td>PolicyTask(PID, TID)</td>
<td>TaskResource (TID, RID)</td>
</tr>
</tbody>
</table>

Table 7. Database schema for a policy base

Figure 19. Entity relationship diagram for process policy
Once the process policies are mapped into database, Visual Editor is used to visualize the process design. Given a process is defined as a directed graph as discussed in Definition 2 in section 4, Visual Editor is implemented on top of Java Universal Network/Graph Framework (JUNG) (http://jung.sourceforge.net/). JUNG is a free open-source software library that provides a common and extendible language for the manipulation, analysis, and visualization of data that can be represented as a graph or network. JUNG is developed in Java, allowing JUNG-based applications to make use of the extensive built-in capabilities of the Java API and other existing third-party Java libraries (Madadhain et al. 2005). We extended JUNG with special extensions for policy-based process design.

As shown in Figure 20, the draft travel reimbursement process depicted in Figure 16

![Figure 20. Screenshot of visual editor](image-url)
is visualized into a directed graph, where different shapes and colors are used to represent tasks, start/end nodes, and decision points. Advanced process analysis, such as adding arcs based on data view, enhancing process using process design rules, is also implemented based on graph theory. Visual Editor also allows process analysts to visually edit process map, e.g., adding/deleting vertices and arcs, which is very convenient and useful for removing semantic errors. Process designer has been tested using the illustrative example presented in section 4 and the travel reimbursement process map was successfully extracted and visualized.

<table>
<thead>
<tr>
<th>Narrative Documentations</th>
<th>Simplified Sentences</th>
<th>Formalized Statements</th>
<th>Logic Formulas</th>
<th>XML Files</th>
</tr>
</thead>
</table>

Figure 21. Process policy representation stack

Process exchanger provides an interface to interoperable with other systems using different process modeling languages. Process policies are represented in different formats as illustrated by the process policy representation stack shown in Figure 21. In particular, process policies are initially embedded in narrative business policy documents, which is then distilled and represented into more precise and succinct sentences according to process policy taxonomy. After that, process policies are formalized using a template and specified as logic formulas using our Unified Predicate Language. However,
UPL is designed as an intra-organizational process modeling language without considering the collaboration among different organizations. Therefore, UPL must be translated into a standard format for process policy exchange and interoperability among different workflow management systems.

Policy $P_4$ in RuleML:

```xml
<Imp>
  <head>
    <Atom>
      <opr><Rel>next</Rel></opr>
      <Ind>Submit TRF</Ind>
      <Ind>Submit TEF</Ind>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <opr><Rel>greater</Rel></opr>
        <Var>reimbursement amount</Var>
        <Ind>limit</Ind>
      </Atom>
    </And>
  </body>
</Imp>
```

Policy $P_5$ in RuleML:

```xml
<Imp>
  <head>
    <Atom>
      <opr><Rel>next</Rel></opr>
      <Ind>Submit TRF</Ind>
      <Ind>TRF Approval</Ind>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <opr><Rel>approved</Rel></opr>
        <Var>TRF</Var>
      </Atom>
    </And>
  </body>
</Imp>
```

Table 8. Process policies represented in RuleML

Process exchanger is able to convert UPL-based process representation into a set of well-known XML-based process modeling standards. In particular, control flow can be converted into BPEL (Andrews et al. 2003) and process constraints can be specified using XACML (OASIS 2003). In addition, Jess-based UPL implementation can be converted into RuleML using the tools provided in SweetRules project (Grosof et al. 2003). As an example, policies “$P_4$: If reimbursement amount exceeds the limit, a Travel Exception Form must be filled” and “$P_5$: After the reimbursement form is approved, a check will be issued” are expressed using RuleML as shown in Table 8. Given RuleML can run on top of
most well-known rule engines such as XSB (http://xsb.sourceforge.net/), Jena (http://jena.sourceforge.net/), IBM CommonRules (http://www.research.ibm.com/rules/),

Besides exporting functions, process exchanger can also import process model represented in BPEL into UPL format, which makes our PPD tool a powerful plug-in to existing BPEL-based workflow engine for process design and change verification.

In this section, we propose an innovative process design approach by means of formal process policy analysis, which is referred as Policy-driven Process Design (PPD). PPD is a new process design methodology different from the existing participative and analytical process design methods. In particular, we formalized the process policy, developed several new concepts such as task view, data view, and process map rules, and designed a set of process design algorithms to facilitate process model extraction. These artifacts build the theoretical foundation for systematic process policy analysis and process design automation.

However, PPD is only discussed by means of representing process model graphically in UML, which is lack of analytical capability. Given different process components including control flow, data flow, resources, and constraints are all extracted from narrative process policies, we propose a process analysis and design language to unify their representation, which we refer to as Unified Predicate Language (UPL). UPL is grounded in predicate logic, whose declarative nature makes it a perfect language to represent policy-based process components. In addition, UPL can facilitate process design by automating several steps discussed in this section via logic inference and queries. UPL is defined and discussed in the next section.
5. A UNIFIED PREDICATE LANGUAGE

Driven by the needs for formally representing process models from a policy perspective and analyzing process changes among different workflow components, we propose a logic-based language for process analysis and design named Unified Predicate Language (UPL). UPL is able to represent control flow, data flow, organizational model, and constraint in a unified logic form and provides various process analysis capabilities based on logic inference and queries. After formally defining UPL, we apply it to process design to demonstrate its power and validate it via a comparison with other two well-know analytical process modeling languages. In section 6, we will use UPL to analyze process changes.

5.1 Language Definition

In this section we specify our Unified Predicate Language (UPL) by defining its variables, predicates and functions. Then, we define the rules that can be expressed in the language. We define the sets of variable symbols as follows:

- \( T = \{t_1, \ldots, t_n, s, e\} \) is the set of tasks, which includes two special tasks: \( s \) (the start node of a workflow) and \( e \) (the end node of a workflow);
- \( RC = \{f, j, d, m\} \) is the set of routing constructs. \( f \) (fork) and \( j \) (join) are used to represent parallel execution and synchronization of a set of task. \( d \) (decision) and \( m \) (merge) are used to choose one task from a set of tasks for execution based on routing constraints.
• $D = \{d_1, \ldots, d_j\}$ is the set of data.

• $U = \{u_1, \ldots, u_k\}$ is the set of users. Users can be human or machine agents.

• $R = \{r_1, \ldots, r_l\}$ is the set of roles, which represents sets of organizational responsibilities that can be assigned to a user, such as department head, account manager, etc. Note that a role can have multiple users and a user can play multiple roles.

• $G = \{g_1, \ldots, g_m\}$ is the set of organizational groups or functional units, such as Bursar’s Office, Graduate College, MIS departments, etc.

• $C = \{c_1, \ldots, c_n\}$ is the set of process constraints, which includes routing constraints, assignment constraints, authorization constraints, etc. As we will discuss later in this section, each constraint is a rule defined in our language.

These variables represent all fundamental elements of a process model, which are used next along with a set of predicates to model different process perspectives.

<table>
<thead>
<tr>
<th>Control Flow Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next$(t_i, t_j)$, next$(t_i, rc_j)$, next$(rc_i, t_j)$, next$(rc_i, rc_j)$</td>
<td>Sequential execution order among tasks and routing constructs.</td>
</tr>
<tr>
<td>fork$(f_i, [t_{j1}, t_{j2}, \ldots, t_{jn}])$</td>
<td>Routing construct $f_i$ leads to a list of parallel task $t_{j1}, t_{j2}, \ldots, t_{jn}$</td>
</tr>
<tr>
<td>join$([t_{i1}, t_{i2}, \ldots, t_{in}], j_j)$</td>
<td>A list of parallel tasks converges to routing construct $j_j$</td>
</tr>
<tr>
<td>branch$(b_i, [t_{j1}, t_{j2}, \ldots, t_{jn}])$</td>
<td>After routing construct $d_i$, exactly one of the tasks in $t_{j1}, t_{j2}, \ldots, t_{jn}$ is to be executed.</td>
</tr>
<tr>
<td>merge$([t_{i1}, t_{i2}, \ldots, t_{in}], j_j)$</td>
<td>Routing construct $m_j$ is triggered once one of the task in $t_{i1}, t_{i2}, \ldots, t_{in}$ completes</td>
</tr>
</tbody>
</table>

Table 9. Control flow predicates

The control flow perspective defines the execution order of tasks. In particular, five
primitive workflow building blocks have been defined as shown in Figure 2 (WfMC 1999b). Accordingly, we propose five predicates to represent the five workflow building blocks as depicted in Table 9.

<table>
<thead>
<tr>
<th>Organization Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role($u_i$, $r_j$)</td>
<td>user $u_i$’s role is $r_j$</td>
</tr>
<tr>
<td>belong($r_i$, $g_j$)</td>
<td>role $r_i$ belongs to organizational group $g_j$</td>
</tr>
<tr>
<td>has($g_i$, $g_j$)</td>
<td>organization group $g_i$ has group $g_j$</td>
</tr>
<tr>
<td>manage($r_i$, $r_j$)</td>
<td>role $r_i$ manages role $r_j$</td>
</tr>
<tr>
<td>manage($u_i$, $u_j$)</td>
<td>user $u_i$ manages user $u_j$</td>
</tr>
<tr>
<td>execute($u_i$, $t_j$)</td>
<td>user $u_i$ is assigned to execute task $t_j$</td>
</tr>
<tr>
<td>execute($r_i$, $t_j$)</td>
<td>role $r_i$ is assigned to execute task $t_j$</td>
</tr>
<tr>
<td>execute($g_i$, $t_j$)</td>
<td>organization group $g_i$ is responsible to handle task $t_j$</td>
</tr>
<tr>
<td>cannot-execute($u_i$, $t_j$)</td>
<td>user $u_i$ cannot be assigned to execute task $t_j$</td>
</tr>
<tr>
<td>cannot-execute($r_i$, $t_j$)</td>
<td>role $r_i$ cannot be assigned to execute task $t_j$</td>
</tr>
<tr>
<td>cannot-execute($g_i$, $t_j$)</td>
<td>organization group $g_i$ cannot be assigned to handle task $t_j$</td>
</tr>
<tr>
<td>constraint($t_i$, $c_j$)</td>
<td>constraint $c_j$ is assigned to task $t_i$</td>
</tr>
<tr>
<td>constraint($b_i$, $c_j$)</td>
<td>constraint $c_j$ is assigned to routing construct branch $b_i$</td>
</tr>
</tbody>
</table>

Table 10. Organization modeling predicates

The organizational perspective defines the hierarchy of organizational resources including users, roles, and organization units. It also specifies their access privileges and responsibilities in terms of tasks they are authorized to handle. In addition, we also capture organizational policies on tasks and routing constructs by specifying their associated constraints if any. For instance, each decision point must have routing constraints to define routing conditions and paths, and temporal policies may restrict certain tasks from execution during non-business hours. Table 10 shows the set of
predicates for modeling the organizational perspective.

The data flow perspective concerns what data is consumed and produced by each task and what are the dependency relationship among data items. Therefore, we specify data flow perspective using the predicates defined in Table 11.

<table>
<thead>
<tr>
<th>Data Flow Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input($t_i$, $d_j$)</td>
<td>data item $d_j$ is the input of task $t_i$</td>
</tr>
<tr>
<td>output($t_i$, $d_j$)</td>
<td>data item $d_j$ is the output of task $t_i$</td>
</tr>
<tr>
<td>depend($d_i$, $d_j$)</td>
<td>data item $d_i$ depends on data item $d_j$</td>
</tr>
</tbody>
</table>

Table 11. Data flow predicates

To facilitate process analysis, we also define a set of comparison functions, two Boolean operation functions, and a set of query functions in Table 12. For example, $\text{count}($execute$(u_i, t_1), n)$ returns the total number of users assigned to task $t_1$ and $\text{findall}($manage$(r_i, r_1), S)$ returns all the roles that have higher level of authorities than role $r_1$ in the organization model.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equal($v_1$, $v_2$), greater($v_1$, $v_2$), greaterequal($v_1$, $v_2$), less($v_1$, $v_2$), lessequal($v_1$, $v_2$)</td>
<td>binary comparison functions corresponding to $=$, $&gt;$, $\geq$, $&lt;$, $\leq$, which return true or false based on comparison results</td>
</tr>
<tr>
<td>plus($v_1$, $v_2$), minus($v_1$, $v_2$), multiply($v_1$, $v_2$), divideby($v_1$, $v_2$)</td>
<td>arithmetic operators corresponding to $+$, $-$, $\times$, $/$</td>
</tr>
<tr>
<td>true($v$), false($v$)</td>
<td>unary function that returns true or false based on the Boolean value of its argument $v$</td>
</tr>
<tr>
<td>count($Q$, $n$)</td>
<td>count the number of different answers of query $Q$ and return the value to $n$</td>
</tr>
<tr>
<td>findall($Q$, $S$)</td>
<td>find all different answers of query $Q$ and return them</td>
</tr>
</tbody>
</table>
to a set $S$

| $\text{findelements}(e, c, E)$ | $e \in \{t, rc, d, u, r, g, c\}$ is a process element. This function returns all process elements of type $e$ involved in constraint $c$ into a set $E$ |
| $\text{findpredicates}(p, c, P)$ | $P$ is one of the control flow, data flow, or organization predicates. This function returns all predicates of type $p$ involved in constraint $c$ into a set $E$ |

Table 12. Auxiliary functions

The rules in our language have the following form:

$$H \leftarrow B_1, B_2, \ldots, B_n, n>0$$

where $H$, $B_1$, $B_2$, \ldots, $B_n$ are predicates or functions. $H$ is the head of the rule and $B_1$, $B_2$, \ldots, $B_n$ consist of the rule body. A rule is triggered if and only if $B_1$, $B_2$, \ldots, $B_n$ are all true, resulting in $H$ inferred to be true. As we mentioned before, process constraints are represented as rules. In particular, we present some build-in rules that are fundamental to process analysis in Table 13.

<table>
<thead>
<tr>
<th>Built-in rules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>research((t_i, t_j)) $\leftarrow$ next((t_i, t_j))</td>
<td>Task reachability rules</td>
</tr>
<tr>
<td>research((t_i, t_j)) $\leftarrow$ next((t_i, f_x)), fork((f_x, [t_{y1}, \ldots, t_{yn}])), member((t_j, [t_{y1}, \ldots, t_{yn}]))</td>
<td></td>
</tr>
<tr>
<td>research((t_i, t_j)) $\leftarrow$ next((j_x, t_j)), join(([t_{y1}, \ldots, t_{yn}], j_x)), member((t_i, [t_{y1}, \ldots, t_{yn}]))</td>
<td></td>
</tr>
<tr>
<td>research((t_i, t_j)) $\leftarrow$ next((t_i, b_x)), branch((b_x, [t_{y1}, \ldots, t_{yn}])), member((t_j, [t_{y1}, \ldots, t_{yn}]))</td>
<td></td>
</tr>
<tr>
<td>research((t_i, t_j)) $\leftarrow$ next((m_x, t_j)), merge(([t_{y1}, \ldots, t_{yn}], m_x)), member((t_i, [t_{y1}, \ldots, t_{yn}]))</td>
<td></td>
</tr>
<tr>
<td>research((t_i, t_j)) $\leftarrow$ next((t_i, t_k)), reach((t_k, t_j))</td>
<td></td>
</tr>
</tbody>
</table>
Table 13. Built-in UPL rules

<table>
<thead>
<tr>
<th>Rule Description</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>User and role hierarchy rules</td>
<td><code>manage(u_i, u_j) ← role(u_i, r_i), role(u_j, r_j), manage(r_i, r_j)</code></td>
</tr>
<tr>
<td></td>
<td><code>manage(r_i, r_j) ← manage(r_i, r_k), manage(r_k, r_j)</code></td>
</tr>
<tr>
<td></td>
<td><code>manage(u_i, u_j) ← manage(u_i, u_k), manage(u_k, u_j)</code></td>
</tr>
<tr>
<td>Authorization inheritance rules</td>
<td><code>execute(u_i, t_j) ← execute(r_k, t_j), role(u_i, r_j)</code></td>
</tr>
<tr>
<td></td>
<td><code>execute(u_i, t_j) ← execute(g_k, t_j), belong(u_i, g_k)</code></td>
</tr>
</tbody>
</table>

Task reachability rules are defined for incomplete path detection. Basically, we say that \( t_j \) is reachable from \( t_i \), if there exists at least one path from \( t_i \) to \( t_j \) in the process model. Intuitively, one task can be reached by another task if they are next to each other or they are directly connected to a routing construct that is of the type Fork, Join, Branch, or Merge.

Task reachability is recursively defined in Table 13, where \( \text{member}(t_i, [t_{i1}, t_{i2}, ..., t_{in}]) \) means that \( t_i \) is a member of the list \([t_{i1}, t_{i2}, ..., t_{in}]\). If the query \( \text{reach}(t_i, t_j) \) returns false, then if \( t_i \) and \( t_j \) are not executed in parallel there is an incomplete path between them. Furthermore, for a correctly design process, \( \text{reach}(s, e) \) should always return true. In this dissertation, we are not focusing on control flow verification and therefore refer interested readers to (Bi et al. 2003a; Bi et al. 2004b; Choi et al. 2002; Sadiq et al. 1996; van der Aalst 1997; van der Aalst 2000b; van der Aalst et al. 2000).

User and role hierarchy rules represent the organization structure and its transitive property. For instance, if \( r_1 \) manages \( r_2 \), and \( r_2 \) manages \( r_3 \), then \( r_1 \) also manages \( r_3 \). These rules can also be used to support dynamic role assignment policies. For example, if a policy states that the role executes task 2 must be the manager of the role that executes task 1. Assume \( r_1 \) executes task 1, and \( r_2 \) is assigned to execute task 2. During runtime, the system conducts a query \( manage(r_2, r_1) \) and only when the query returns true \( r_2 \) is allowed to
execute task 2. Authorization inheritance rules define that users can inherit task execution authorizations from the role and/or organization units they are affiliated with. For example, if an authorization policy says that bursar’s office is in charge of issue reimbursement check, then everyone in bursar’s office is authorized to issue those checks. Authorization inheritance rules are created to support role-based and team-based access control.

With a succinct set of predicates, UPL is expressive enough to model control flow, data flow, organization model, and process constraints. It also provides great extensibility and flexibility via its capability of defining different sets of rules for different process modeling and analysis requirements. Next, we show how our language can help process design.

5.2 Process Design via UPL

In this section, we demonstrate the unique capability of our unified predicate language (UPL) in terms of facilitating process design. Specifically, UPL is able to represent narrative process information in a succinct and accurate form, provide process design insights via reasoning about relationship among process elements, and help detect design errors by conducting queries.

Table 14 shows a set of travel reimbursement process policies we synthesized from a major public university’s business policy manual. In the rest of this section, we illustrate how a travel reimbursement process can be systematically designed from these narrative process policies by applying our Unified Predicate Language (UPL).
P1: All claims for the reimbursement of expenses for an approved university business travel are made on a Travel Reimbursement Form (TRF).

P2: If Travel Reimbursement Form is submitted later than 60 days upon completing the travel, a Reasonable Exception Request Form (RERF) must be submitted.

P3: If reasonable exception request is not approved, then the reimbursement amount will be taxable.

P4: If reimbursement amount exceeds the limit, a Travel Exception Form (TEF) must be filled.

P5: After the reimbursement form is approved, a check will be issued.

P6: Department or unit head must approve the travel.

P7: Department or unit head must approve the reasonable exception request form.

P8: Disbursement Services Center (DSC) must review and approve the reimbursement form.

P9: The Office of Business and Financial Services (OBFS) reviews the travel exception form.

P10: All travel exceptions must be submitted to Higher Education Travel Control Board (HETC) for quarterly review.

P11: Bursar’s Office (BO) is responsible to issue reimbursement check.

Table 14. Travel reimbursement policies

Figure 22. Procedure for process design via UPL
Figure 22 shows the procedure of process design using UPL. This procedure is similar to process extraction procedure presented in Figure 10. However, instead of using graphical modeling language UML and manually go through the procedure, UPL further facilitate process design by recording process design steps in an accurate and computational logic form and automating process design via logic deduction. In addition, we discuss more about UPL-based organization and constraints modeling, which was not covered in the procedure in Figure 10. According to the procedure, we first identify a set of tasks, data, roles, organization units, and routing constructs as shown below:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>t₁(request travel), t₂(approve travel), t₃(submit TRF), t₄(submit RERF), t₅(approve RERF), t₆(make reimbursement taxable), t₇(submit TEF), t₈(approve TEF), t₉(submit TEF for quarterly review), t₁₀(approve reimbursement), t₁₁(issue check)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>d₁(TRF), d₂(RERF), d₃(TEF), d₄(check), d₅(travel completion time), d₆(reimbursement request submission time), d₇(reimbursement amount), d₈(TRF approval), d₉(RERF approval), d₁₀(TEF approval), d₁₁(check), d₁₂(travel amount limit), d₁₃(travel request), d₁₄(travel approval)</td>
</tr>
<tr>
<td>Roles</td>
<td>r₁(department/unit head)</td>
</tr>
<tr>
<td>Organization Units</td>
<td>g₁(Disbursement Services Center), g₂(Office of Business and Financial Services), g₃(Higher Education Travel Control Board), g₄(Bursar’s Office)</td>
</tr>
<tr>
<td>Routing Constructs</td>
<td>b₁(whether TRF is submitted after 60 days of travel completion date), b₂(whether reimbursement amount exceeds the limit), b₃(whether RERF is approved), b₄(whether TEF is approved), b₅(whether TRF is approved)</td>
</tr>
</tbody>
</table>

Table 15. Identified process elements from process policies

We can also obtain control flow information from the policies. For instance, policy P3 implies that based on the decision (b₃) of reasonable exception approval (t₃), the reimbursement amount should be made taxable (t₆), which can be precisely represented in
UPL as $\text{branch}(b_3, [t_6]), \text{next}(t_5, b_3)$. Similarly, we can get other control flow information as follows: $\text{branch}(b_1, [t_4]), \text{branch}(b_3, [t_6]), \text{next}(t_5, b_3), \text{branch}(b_2, [t_7]), \text{branch}(b_4, [t_9]), \text{next}(t_8, b_4), \text{branch}(b_5, [t_{11}]), \text{next}(t_{10}, b_5)$. However, these control flow information directly retrieved from process policies contains only limited information on the whole travel reimbursement process. Therefore, more information on the execution order among tasks needs to be added.

Each task in a process model is associated with a set of input and output data. There is also dependency relationship among data items, e.g., the Travel Exception Form (TEF) depends on the Travel Reimbursement Form (TRF), because only after TRF is submitted, TEF can be created. Data flow based workflow design has been studied in (Sun et al. 2004a). Intuitively, if there is a data dependency among the sets of data items associated with two tasks, those two tasks should be executed sequentially; otherwise, they can be executed in parallel. In UPL, we can define the data dependency rules as follows:

| $\text{next}(t_i, t_j) \leftarrow \text{output}(t_i, d_k)$, $\text{input}(t_j, d_k)$ | $t_i$ is executed before $t_j$ if $t_i$ outputs $d_k$ that is $t_j$’s input |
| $t_i$ is executed before $t_j$ if $t_i$ outputs $d_m$, $t_j$ inputs $d_m$, and $d_n$ depends on $d_m$ |

**Table 16. Data dependency rules**

| inputs and outputs: $\text{output}(t_1, d_{13}), \text{input}(t_2, d_{13}), \text{output}(t_2, d_{14}), \text{output}(t_3, d_1), \text{output}(t_3, d_5), \text{output}(t_3, d_6), \text{output}(t_3, d_7), \text{output}(t_4, d_2), \text{input}(t_5, d_2), \text{output}(t_5, d_{10}), \text{input}(t_6, d_7), \text{output}(t_7, d_3), \text{input}(t_8, d_3), \text{output}(t_8, d_{11}), \text{input}(t_9, d_3), \text{input}(t_{10}, d_1), \text{output}(t_{10}, d_9), \text{input}(t_{11}, d_9), \text{input}(t_{11}, d_7), \text{output}(t_{11}, d_4), \text{input}(b_1, d_5), \text{input}(b_1, d_6), \text{input}(b_2, d_7), \text{input}(b_2, d_{12}), \text{input}(b_3, d_2), \text{input}(b_4, d_3), \text{input}(b_5, d_1)$ |

**Table 17. Data inputs and outputs**
In our example, the inputs and outputs for each task are shown in Table 17. By applying data dependency rules, additional control flow information can be inferred as follows: \( \text{next}(t_1, t_2), \text{next}(t_3, t_{10}), \text{next}(t_3, t_{11}), \text{next}(t_4, t_5), \text{next}(t_7, t_8), \text{next}(t_7, t_9) \). These new control flow information need to be checked against process policies to ensure they are semantically right. Based on additional policy analysis, we can draft a control flow diagram as shown in Figure 23. According to control flow information we get directly from process policies and information in Figure 23, we can decide that \( \text{next}(t_3, t_{10}), \text{next}(t_3, t_{11}), \text{next}(t_7, t_9) \) should be removed and \( \text{next}(t_2, t_3), \text{next}(t_3, f_1), \text{fork}(f_1, [b_1, b_2]), \text{join}([m_1, m_2], j_1), \text{next}(j_1, t_{10}), \text{next}(s, t_1) \) should be added. So far, all control flow information are listed in Table 18.

\[
\begin{align*}
\text{next}(s, t_1), & \quad \text{next}(t_1, t_2), \quad \text{next}(t_2, t_3), \quad \text{next}(t_3, f_1), \quad \text{fork}(f_1, [b_1, b_2]), \quad \text{decision}(b_1, [t_4]), \quad \text{next}(t_4, t_5), \\
\text{next}(t_5, b_3), & \quad \text{decision}(b_3, [t_6]), \quad \text{decision}(b_2, [t_7]), \quad \text{next}(t_7, t_8), \quad \text{next}(t_8, d_4), \quad \text{decision}(b_4, [t_9]), \\
\text{join}([m_1, m_2], j_1), & \quad \text{next}(j_1, t_{10}), \quad \text{next}(t_{10}, b_3), \quad \text{decision}(b_5, [t_{11}])
\end{align*}
\]

Table 18. Partial control flow information

Now we want to know whether the control flow is correct. Next, we first define the syntactical correctness for control flow modeling and then demonstrate how UPL can help complete control flow design.
**Definition 4 (Syntactical Correctness of Control Flow Design).** A control flow diagram is correct if the following conditions hold

1. there is one and only one outgoing link for the start node and one and only one incoming link for the end node;
2. there is one and only one incoming and outgoing link for each task;
3. there must be one and only one incoming link and at least two outing links for routing construct *fork* and *branch*;
4. there must be at least two incoming links and one and only one outing link for routing construct *join* and *merge*.

Note that Definition 4 releases some assumptions we made about a process model syntactical correctness as defined in Definition 3. In particular, UPL provides predicates to model parallelism, which was not available in process map definition. Once the control flow information is represented using UPL as in Table 18, we can use auxiliary functions in Table 12 to quickly check its syntactical correctness. In particular, an algorithm for process syntactical correctness checking is developed as shown in Table 19. We use ‘\_\_’ to represent a single logic symbol and use ‘\&\_\_’ to represent any number of symbols. For instance, \(\text{count}(\text{next}(s, \_), n)\) returns \(n\) as the number of outgoing links for the start node.

If \(n \neq 1\), then \(s\) is identified to be incorrect for control flow design. Similarly, for task \(t_i\), \(\text{count}(\text{next}(\_, t_i), x_1), \text{count}(\text{fork}(\_, [\&\_\_, t_i]), x_2), \text{count}(\text{branch}(\_, [\&\_\_, t_i]), x_3)\) return the number of possible incoming links to \(x_1, x_2,\) and \(x_3\), and \(t_i\) is considered correctly designed only when \((x_1 + x_2 + x_3) = 1\).

<table>
<thead>
<tr>
<th><strong>INPUT:</strong></th>
<th>control flow information in UPL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OUTPUT:</strong></td>
<td>1. TRUE if the control flow is syntactical correct, otherwise</td>
</tr>
<tr>
<td></td>
<td>2. FALSE with identified problems</td>
</tr>
</tbody>
</table>
**Procedure** ControlFlowSyntacticalCorrectnessVerification

BEGIN

//check rule 1

`count(next(s, _), n); count(next(_, e), m);`

if `n ≠ 1`

`return FALSE and s;`

`exit;`

end if

if `m ≠ 1`

`return FALSE and e;`

`exit;`

end if

// check rule 2

for each task `t`

`count(next(_, t), x1); count(fork(_, [&_, t]), x2); count(branch(_, [&_, t]), x3);`

`count(next(ti, _), y1); count(join([&_, t], _), y2); count(merge([&_, t], _), y3);`

if `(x1+x2+x3) ≠ 1`

`return FALSE and t;`

`exit;`

end if

if `(y1+y2+y3) ≠ 1`

`return FALSE and t;`

`exit;`

end if

end for

//check rule 3 and 4

for each routing construct `f, b, j, m`

`count(next(_, f), x1); count(fork(f, [_ , _ , &]), x2);`

`count(next(_, b), x3); count(branch(b, [_ , _ , &]), x4);`

`count(next(_, j), x5); count(join([& _, _], j), x6);`

`count(next(_, m), x7); count(merge([& _, _], m), x8);`

if `(x1≠1 or x2≠1)`

`return FALSE and f;`

`exit;`

end if

if `(x3≠1 or x4≠1)`

`return FALSE and b;`

`exit;`

end if

if `(x5≠1 or x6≠1)`

`return FALSE and j;`

`exit;`

end if

if `(x7≠1 or x8≠1)`

```
return FALSE and $m$;
exit;
end if
end for
END

Table 19. Algorithm for syntactical verification of control flows

By running the verification algorithm on facts in Table 18, we are able to identify control flow design problems for the travel reimbursement process as shown in Table 20. Without modeling process design in UPL, the verification of control flow syntactical correctness is largely based on human inspection. Human inspection is error-prone and very inefficient when the control flow is complicated. UPL can greatly improve process design efficiency and correctness by transforming graphical process representation into rigorous logic format and detecting design problems via automatic logic reasoning.

<table>
<thead>
<tr>
<th>Identified problems</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_1, b_2, b_3, b_4, b_5$</td>
<td>these decision points only have one outgoing link</td>
</tr>
<tr>
<td>$m_1, m_2$</td>
<td>these merges have no incoming link</td>
</tr>
<tr>
<td>$t_{11}, e$</td>
<td>$t_{11}$ has no outgoing link and $e$ has no incoming link</td>
</tr>
</tbody>
</table>

Table 20. Identified design problems

Table 21 shows the final control flow information after correcting identified design problem, which corresponds to the travel reimbursement process diagram in Figure 24.

Table 21. Complete control flow information
After the control flow design is complete, the organizational design of the process model should be conducted including identifying process related resources, such as users, role, and organization units, assigning resources to tasks, capturing process constraints, and assigning constraints to proper routing constructs. In our example, we have identified one role and four organization units from process policies as shown in Table 15. Process policies \( p_6, p_7, p_8, p_9, \) and \( p_{11} \) explicitly state how the identified resources are assigned to different tasks, which can be represented in UPL as follows: \( \text{execute}(r_1, t_2), \text{execute}(r_1, t_3), \text{execute}(g_1, t_{10}), \text{execute}(g_2, t_8), \text{execute}(g_4, t_{11}). \) In order to make a process executable, each task must be assigned to some resources for execution. The query \( \text{count}(\text{execute}(\_ , t_i), n) \) returns the number of resources assigned to a task \( t_i \), which enables us to easily identify tasks without resources where \( n \) equals to 0.

Figure 24. Control flow of travel reimbursement process
For each routing construct branch, there must be constraints to define how to route process instance under different conditions. In our example, we have process policies p2, p3, p4, p5, p8 in Table 14 defining the routing constraints, which can be expressed in UPL as shown in Table 22. Then, we assign those constraints to different routing constructs as follows: constraint(b1, c1), constraint(b1, c2), constraint(b2, c3), constraint(b2, c4), constraint(b3, c5), constraint(b3, c6), constraint(b4, c7), constraint(b4, c8), constraint(b5, c9), constraint(b5, c10). For each decision point, there must be at least two constraints to control the routing. The query count(constraint(bi, _), n) returns the number of constraints associated with branch bi. When the query result n is less than 2, additional constraints need to be added to corresponding routing constructs.

In this section, we demonstrated that our Unified Predicate Language can record process design in a succinct and accurate manner. It can also provide process design insights and identify design problems via logic reference and queries. Next, we further validate UPL by comparing it with other process modeling languages.

5.3 Comparison with other languages

In this section, we compare our proposed Unified Predicate Language (UPL) with other two well-know process modeling and analysis approaches, namely, Petri nets (van
der Aalst 1998) and Metagraphs (Basu et al. 2000). Note that we exclude graphical languages such as UML (OMG 2003), IDEF3 (Mayer et al. 1995), and BPMN (OMG 2006) that have few analytical capability, execution languages such as BPEL (Andrews et al. 2003) and WSCI (Arkin et al. 2002), and languages used to specifically model workflow authorization constraints and exceptions (Bertino et al. 1999; Casati et al. 1999). The comparison result is shown in Table 23, where ‘n/a’ means that the information is not found in the referenced papers.

Compared with the other two approaches, UPL has richer expressiveness, because it is the only approach that can represent all four major process perspectives, namely, control flow, data flow, organization model, and process constraints. In particular, Petri nets focus mainly on control flow aspect and Metagraphs totally ignore process constraint modeling. UPL is able to facilitate process design by providing design insights and identifying design problems via logic inferences and queries, which cannot be handled by other two approaches. In addition, as we will discuss in the next section, the unified logic representation of different process components also endorses UPL the capability to analyze process constraint changes, which is also missing from Petri nets and Metagraphs methods.

In terms of tool support, Petri nets based workflow modeling and analysis tools are limited and to the best of our knowledge, there is no Metagraphs based tools reported in the literature. UPL is grounded in first-order logic and its syntax is similar to the well-known logic programming language Prolog (Sterling et al. 1994). There has been many research and implementations of rule engines, such as JESS and XSB, which enable UPL to take advantage of those rule engines for execution. Although by itself UPL has no graphical
interface, due to the clearly defined process modeling predicates with formal semantics, visual representation for UPL can be easily developed.

<table>
<thead>
<tr>
<th>Control Flow Modeling</th>
<th>UPL</th>
<th>Petri nets</th>
<th>Metagraphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Sequential</td>
<td>● Sequential</td>
<td>● Sequential</td>
<td>● Sequential</td>
</tr>
<tr>
<td>● Fork/Join</td>
<td>● Fork/Join</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Branch/Merge</td>
<td>● Branch/Merge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Iteration</td>
<td>● Iteration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Flow Modeling</td>
<td>● Data dependency</td>
<td>n/a</td>
<td>Data-task interaction</td>
</tr>
<tr>
<td>● Data-task interaction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization Modeling</td>
<td>● Users, roles, groups</td>
<td>n/a</td>
<td>roles</td>
</tr>
<tr>
<td>● Role hierarchy</td>
<td>● Role hierarchy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Authorization</td>
<td>● Authorization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Inheritance</td>
<td>● Inheritance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Constraint Modeling</td>
<td>● Assignment constraint</td>
<td>Routing constraint</td>
<td>n/a</td>
</tr>
<tr>
<td>● Routing constraint</td>
<td>● Routing constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Policy constraint</td>
<td>● Policy constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical Representation</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Process Design Facilitation</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Constraint Change Analysis</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Tool support</td>
<td>most rule engines, e.g., JESS, XSB, etc.</td>
<td>Woflan</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 23. Comparisons of three process modeling and analysis approaches

In this section, we proposed a Unified Predicate Language (UPL) to unify the representation of different process perspectives. We demonstrated power of UPL by applying it to process design. We also illustrated the advantages of UPL over other process modeling languages through a comparison. Besides all the features we have discussed so far, UPL also enables the change analysis among different process perspectives as we will present next.
6. PROCESS CHANGE ANALYSIS

To be successful and competitive, organizations need to quickly adapt their business processes to changes in the business environment, such as mergers/acquisitions, new regulations and customer demand, and advance of technology. For organizations with large scale business processes, maintaining process consistency under various changes is a very challenging task. Although process changes have been discussed in adaptive and flexible workflow research area, most existing discussions are usually focusing on changes within one process perspective, such as control flow (Kammer et al. 2000; Sadiq et al. 2000; van der Aalst 2001), data flow (Sadiq et al. 2004), and process constraints (Casati et al. 2001; Casati et al. 1999). Little research has been done on the analysis of change relationship among different process perspectives. For instance, questions like “how the remodeling of organization structure due to the merger can affect our existing processes?” and “what are the consequences of removing these tasks in terms of process consistency?” cannot be easily answered without a systematic mechanism of process change analysis. Consequently, most existing business process management systems lack process change analysis capability and cannot support efficient and flexible process adaptation.

In this section, we first summarize different types of changes into three basic operations on different process components, i.e., insertion, deletion, and update. Then, we discuss process change relationship and focus on how changes in control flow and organization model can affect process constraint. We use UPL to formally define the
different types of process constraint anomalies due to process changes and develop algorithms to verify and enforce process consistency. The algorithm is validated via an example. Next, we first describe an example process we will in the rest of this section.

6.1 A Procurement Approval Process

A procurement approval workflow is represented using an UML activity diagram in Figure 25, and each task is described in Table 24. Basically, this process consists of two major parts: procurement request approval and order processing.

<table>
<thead>
<tr>
<th>Task</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Purchase Request</td>
<td>Employees submit purchase request.</td>
</tr>
<tr>
<td>t2</td>
<td>Supervisor Approval</td>
<td>The supervisor of task initiator approves the request</td>
</tr>
</tbody>
</table>

Figure 25. Procurement process in UML
When request amount is greater than 1000, a checking is conducted to make sure there is sufficient fund.

When fund is sufficient, higher level manager needs to approve.

Purchase Request is reviewed and sent to inventory. PO is requested to be printed.

Inventory checking. If in stock, start shipping; otherwise back order.

PO is printed.

The requested item is back ordered.

Proper department is billed for the items ordered.

Item is shipped.

The purchase transaction information is summarized and archived.

The requestor is notified that his/her purchase request is declined via email.

The control flow can be represented using UPL as shown below:

```
next(s, t1), next(t1, t2), next(t2, b1), branch(b1, [b2, t12]), branch(b2, [t3, t5]), next(t3, b3),
branch(b3, [t4, t12]), next(t4, b4), branch(b4, [t5, t12]), next(t5, t7), next(t7, b5), branch(b5, [t8, f1]),
next(t8, b6), branch(b6, [t7, m1]), fork(f1, [t6, t9]), next(t9, t10), join([t6, t10], j1),
merge([j1, b6], m1), next(m1, t11), merge([t11, t12], m2), next(m2, e)
```

The data items in the procurement approval process are summarized in Table 26. The role hierarchy associated with the workflow is presented in Figure 26. The employee appointment information (user-role assignment) is as follows: GM: {John}; ED: {Joe}; PD: {Jason}; AD: {Maggie}; SE: {Eric, Ray}; PC: {Peter}; IC: {Sam}; DC: {Dan, Jack};
AC: {Steve, Ben}.

d₁: purchase request; d₂: supervisor approval; d₃: purchase amount; d₄: fund amount; d₅: mgmt approval; d₆: requested item amount; d₇: requested item inventory; d₈: back order time; d₉: item receiving time; d₁₀: purchase order; d₁₁: billing information; d₁₂: shipping information; d₁₃: purchase archive; d₁₄: disapproval notice

Table 26. Data items in procurement process

![Organization model for the procurement process](image)

The organizational model in Figure 26 can be expressed as the logical statements in Table 27.

manage(GM, ED), manage(GM, PD), manage(GM, AD), manage(ED, SE), manage(PD, PC), manage(PD, IC), manage(PD, DC), manage(AD, AC), role(John, GM), role(Joe, ED), role(Jason, PD), role(Maggie, AD), role(Eric, SE), role(Ray, SE), role(Peter, PC), role(Sam, IC), role(Jack, DC), role(Dan, DC), role(Steve, AC), role(Ben, AC)

Table 27. Organization structure in UPL

Given the build-in rules on role hierarchy and authorization inheritance shown in Table 13, we can retrieve useful information by querying the organizational facts. For instance, query manage(_, Eric) shows the supervisor of Eric, which is Joe in this case and query manage(Jason, Eric) returns False, because Jason is not the manager of Eric. These
queries are used to implement runtime user or role resolution and can also be utilized to analyze organizational structure changes.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>t2 (Supervisor Approval) must be executed by the supervisor of the purchase requestor.</td>
</tr>
<tr>
<td>c2</td>
<td>t3 (Fund Checking) must be executed by an Accounting Clerk (AC).</td>
</tr>
<tr>
<td>c3</td>
<td>The role associated with t4 (Mgmt. Approval) must be the supervisor of the role that executes t2.</td>
</tr>
<tr>
<td>c4</td>
<td>t5 (Purchasing) is handled by a Purchasing Clerk (PC)</td>
</tr>
<tr>
<td>c5</td>
<td>t6 (Inventory Checking) is conducted by an Inventory Clerk (IC).</td>
</tr>
<tr>
<td>c6</td>
<td>t9 (Billing) must be handled by a Distribution Clerk (DC).</td>
</tr>
<tr>
<td>c7</td>
<td>t10 (Shipping) must be handled by a Distribution Clerk (DC).</td>
</tr>
<tr>
<td>c8</td>
<td>t11 (Archiving) is handled by a Purchasing Clerk (PC).</td>
</tr>
<tr>
<td>c9</td>
<td>t1 (Purchase Request) and t3 (Fund Checking) cannot be done by the same person.</td>
</tr>
<tr>
<td>c10</td>
<td>t5 (Billing) and t10 (Shipping) must be handled by the same person.</td>
</tr>
<tr>
<td>c11</td>
<td>if supervisor approves the request, b2 will be activated.</td>
</tr>
<tr>
<td>c12</td>
<td>if supervisor does not approve the request, t12 (disapproval notification) is activated.</td>
</tr>
<tr>
<td>c13</td>
<td>if request amount is greater than one thousand, t3 (Fund Checking) is executed.</td>
</tr>
<tr>
<td>c14</td>
<td>if request amount is less than one thousand, t5 (Purchasing) is executed.</td>
</tr>
<tr>
<td>c15</td>
<td>if there is enough fund, then t4 (Mgmt Approval) is executed.</td>
</tr>
<tr>
<td>c16</td>
<td>if there is not enough fund, t12 (disapproval notification) is activated.</td>
</tr>
<tr>
<td>c17</td>
<td>if management approves the request, t5 (Purchasing) is executed.</td>
</tr>
<tr>
<td>c18</td>
<td>if management does not approve the request, t12 (disapproval notification) is activated.</td>
</tr>
<tr>
<td>c19</td>
<td>if requested items are out of stock, t8 (Back Order) is executed.</td>
</tr>
<tr>
<td>c20</td>
<td>if requested items are in stock, a set of parallel order processing tasks are executed.</td>
</tr>
<tr>
<td>c21</td>
<td>if items are received within 6 days after back ordering, inventory should be updated and rechecked.</td>
</tr>
<tr>
<td>c22</td>
<td>if items are not received 6 days after being back-ordered, the request should be archived and completed.</td>
</tr>
</tbody>
</table>

Table 28. Constraints of requisition approval process
To ensure the proper execution of the process, roles and users may be assigned to tasks and process constraints should be specified to enforce business requirements. Table 28 shows the constraints for the requisition approval process. Constraints $c_1$ through $c_8$ are role-task assignments. Constraint $c_9$ is an instance of separation of duties policy, whereas $c_{10}$ is a binding of duties policy. Constraints $c_{11}$ through $c_{22}$ are routing rules for the six decision points in the process. In particular, $c_1$, $c_3$, $c_9$, $c_{10}$, and $c_{11}$ to $c_{22}$ are dynamic constraints, which can be evaluated only during process execution. These constraints can be precisely represented in UPL as follows:

<table>
<thead>
<tr>
<th>Constraint</th>
<th>UPL Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$: execute($r_i$, $t_2$) ← manage($r_i$, $r_j$), execute($r_i$, $t_1$)</td>
<td>$c_{11}$: next($b_1$, $b_2$) ← true($d_3$)</td>
</tr>
<tr>
<td>$c_2$: execute($AC$, $t_3$)</td>
<td>$c_{12}$: next($b_1$, $t_{12}$) ← false($d_3$)</td>
</tr>
<tr>
<td>$c_3$: manage($r_i$, $r_j$) ← execute($r_i$, $t_4$), execute($r_j$, $t_2$)</td>
<td>$c_{13}$: next($b_2$, $t_3$) ← greaterequal($d_3$, 1000)</td>
</tr>
<tr>
<td>$c_4$: execute($PC$, $t_5$)</td>
<td>$c_{14}$: next($b_2$, $t_5$) ← less($d_3$, 1000)</td>
</tr>
<tr>
<td>$c_5$: execute($IC$, $t_6$)</td>
<td>$c_{15}$: next($b_3$, $t_4$) ← greaterequal($d_3$, $d_3$, 0)</td>
</tr>
<tr>
<td>$c_6$: execute($DC$, $t_9$)</td>
<td>$c_{16}$: next($b_3$, $t_{12}$) ← less($d_3$, $d_3$, 0)</td>
</tr>
<tr>
<td>$c_7$: execute($DC$, $t_{10}$)</td>
<td>$c_{17}$: next($b_4$, $t_3$) ← true($d_3$)</td>
</tr>
<tr>
<td>$c_8$: execute($PC$, $t_{12}$)</td>
<td>$c_{18}$: next($b_4$, $t_3$) ← false($d_3$)</td>
</tr>
<tr>
<td>$c_9$: cannot_execute($u_i$, $t_3$) ← execute($u_i$, $t_4$)</td>
<td>$c_{19}$: next($b_5$, $t_8$) ← greater($d_6$, $d_7$)</td>
</tr>
<tr>
<td>$c_{10}$: execute($u_i$, $t_{10}$) ← execute($u_i$, $t_9$)</td>
<td>$c_{20}$: next($b_5$, $f_1$) ← lessequal($d_6$, $d_7$)</td>
</tr>
<tr>
<td></td>
<td>$c_{21}$: next($b_6$, $t_7$) ← lessequal($d_9$, $d_8$, 6)</td>
</tr>
<tr>
<td></td>
<td>$c_{22}$: next($b_6$, $m_1$) ← greater($d_9$, $d_8$, 6)</td>
</tr>
</tbody>
</table>

Table 29. Process Constraints in UPL

Given the UPL representation of the workflow model, we will show later in this section that process change analysis can be carried out via logic inference and queries. Next, we first study the types of process changes and change relationship among different process components.
6.2 Process Change Types and Relationship

<table>
<thead>
<tr>
<th></th>
<th>Insertion</th>
<th>Deletion</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control flow</td>
<td>● Adding a task</td>
<td>● Remove a task</td>
<td>● Change task execution order</td>
</tr>
<tr>
<td></td>
<td>● Adding a routing construct</td>
<td>● Remove a routing construct</td>
<td>● Change sequential execution into parallel execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td>● Hire a employee</td>
<td>● Employee leaves organization</td>
<td>● Reassign employee to a new position</td>
</tr>
<tr>
<td></td>
<td>● Create a new position (role)</td>
<td>● Remove a position</td>
<td>● Reorganize the members of an organization unit</td>
</tr>
<tr>
<td></td>
<td>● Create an organization unit</td>
<td>● Dismiss an organization unit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data flow</td>
<td>● Adding new data item</td>
<td>● Remove data item</td>
<td>● Change data item value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraints</td>
<td>● Assign more resources to task</td>
<td>● Remove resources from task execution</td>
<td>● Revise routing conditions</td>
</tr>
<tr>
<td></td>
<td>● Assign more constraints to routing constructs</td>
<td>● Remove routing constraints</td>
<td>● Revise process execution rules</td>
</tr>
<tr>
<td></td>
<td>● Define more process execution rules,</td>
<td>● Remove process execution rules</td>
<td>such as temporal constraints, dynamic authorization constraints,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exception handling constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 30. Classification of process changes

A process model has four major components, namely, control flow, data flow, organizational model, and constraints. Different changes can occur to any one of those components, e.g., altering task execution sequence, relocating organization resources, adding constraints for regulation compliance, etc. Although different changes have different semantics, all those changes can be classified into three basic operations from database implementation perspective, i.e., Insertion, Deletion, and Update. For example, adding a task is an insertion change to the control flow, removing a role is a deletion change
to the organizational model, and redefining the routing conditions for a decision point is an update change to process constraints. Table 30 demonstrates some typical process changes.

Given that different process components interact with one another in a process model, changes in one component can have significant consequences in other components. For instance, process redesign that removes a set of non-value-added tasks may lead to organization downsizing, and process policy changes may force adding additional tasks. For organizations with large scale business processes, identifying change consequences to different process components in an efficient way is a very challenging task. Figure 27 illustrates the change relationships among different process component, which are discuss in more detail as follows:

Figure 27. Change relationships among process components

- Change relationship 1. The changes in the control flow can greatly affect process constraint consistency. For example, we have a separation of duty constraint saying “Purchase request and fund checking cannot be done by the same person”. If the task fund checking is removed due to process redesign, the constraint will never be violated.
However, if the constraint is not removed or inactivated, the workflow system will still keep evaluating and enforcing this constraint resulting in waste of system resources and inefficiency.

- Change relationship 2. Data is produced and consumed by tasks in the control flow. Data dependency and anomalies have been studied in (Sadiq et al. 2004; Sun et al. 2004b). The changes in control flow may lead to data flow anomalies, which can cause runtime data access errors or data inconsistency. For instance, if a task $t_1$ consumes a data item produced by $t_2$, a missing data error will occur if $t_2$ is removed.

- Change relationship 3. Some process constraints are defined based on data item values. For example, a routing constraint may define that “if the credit score of the applicant is above 750, then a special APR is applied”. This constraint relies on the credit score for its evaluation. If this credit score data item is not produced by tasks before this constraint is evaluated, a runtime error occurs.

- Change relationship 4. Processes are running in an organization setting. When organization changes, processes are also subject to change. For instance, mergers/acquisitions may result in integrating business processes from different organizations, in which the control flow must be refined to best serve the new organization’s structure.

- Change relationship 5. A major set of process constraints are related to resource-task assignment. When some organizational resources become unavailable, assignment constraints must be revised to avoid runtime resource resolution errors. For instance, a constraint can specify that “the inventory clerk is responsible for back order all items...”
that are out of stock”. This constraint assigns a role “inventory clerk” to a task “back ordering”. When the role “inventory clerk” is unavailable, either the role is removed or no employee is currently assigned to the role, another user or role must be delegated to handle the task. Otherwise, the process is stalled resulting in long cycle time and low customer satisfaction.

Analyzing each of the changes aforementioned in detail is not the focus of this dissertation. Instead, we aim to show how process changes among different process components can be formally analyzed using our proposed unified predicate language. Given process constraints contains information on all other three process components, we emphasize on how the changes in control flow and organization model can affect process constraints, i.e., change relationship 1 and 5. Next, we first investigate a set of potential process constraint anomalies due to process changes.

6.3 Process Constraint Anomalies

Process constraints are business rules defined to control and restrict process execution. We can classify process constraints into three categories, namely, Routing Constraints, Assignment Constraints, and Policy Constraints. Routing constraints defines the routing conditions for each decision point in the control flow. Assignment constraints define who executes each task and what routing constraints are related to each routing construct. Authority constraints represent the organizational structure and use-role relationship. Policy constraints specify the rules that the process execution must follow, e.g., a separation of duty policy may state that an employee cannot approve the procurement
request submitted by himself, and a temporal policy may require a credit approval task to complete within three days.

Process constraints can involve information on control flow, data flow, and organization, which includes all process elements, i.e., task, routing construct, data, user, role, and organization units. Therefore, when changes happen to those process elements, process constraints must be verified for correctness and consistency, otherwise process constraint anomalies can occur. In particular, process constraint anomalies can be classified into four categories, namely, *Missing Constraints*, *Redundant Constraint*, *Invalid Constraint*, and *Conflicting Constraint*, which are described in detail as follows:

Missing constraint anomaly occurs when a process cannot be properly executed due to the lack of assignment constraints. The following scenarios can cause missing constraint anomaly.

**Scenario 1.** A task has no resource associated with it for execution.

**Scenario 2.** A decision point has no constraint associated with it to specify routing conditions.

Constraints are identified as redundant if they are never evaluated in process execution. Redundant constraints are usually caused by the deletion of tasks and routing constructs and can result in inefficiency and waste of resources. The following scenarios can cause redundant constraint anomaly.

**Scenario 3.** All tasks and routing constructs in the constraint do not exist. For example, constraint $c_8$ says that $t_{11}$ (Archiving) is handled by a Purchasing Clerk (PC). If $t_{11}$ is removed, $c_8$ becomes a redundant constraint although role ‘PC’ still exists, because $c_8$ is
never evaluated.

Invalid constraints contain erroneous information, which leads to runtime process execution exceptions. The following scenarios can cause invalid constraints.

**Scenario 4.** Resources or constraints specified in assignment constraints do not exist. For example, constraint c7 says t_{10} (Shipping) must be handled by a Distribution Clerk (DC). If role ‘DC’ is not available, c_{7} becomes an invalid constraint that leads to a runtime role resolution exception.

**Scenario 5.** Task information in routing constraints does not exist. For example, routing constraint c_{13} defines that if request amount is greater than one thousand, t_{3} (Fund Checking) is executed. If t_{3} is deleted, c_{13} becomes an invalid resulting in a runtime routing exception.

Conflicting constraint anomaly occurs when there are constraints that contradict with one another preventing process from normal execution. The following scenarios can cause conflicting constraint anomaly.

**Scenario 6.** There are conflicting assignment constraints. For instance, if there is one constraint specifying a role r_{i} can execute task t_{j}, while another constraint defines that r_{i} cannot execute task t_{j}, a contradiction is reached.

**Scenario 7.** There are conflicting routing constraints. For instance, if there is one constraint specifying task t_{m} is executed after task t_{n}, while another constraint defines that t_{k} is executed after task t_{n}, a routing confliction happens.

Given the four types of process constraint anomalies, the process constraints must be verified when changes happen to process model. Proper actions must also be taken
according to the anomaly types to remove inconsistency. Specifically, additional
constraints need to be added to remove missing constraint anomaly. Redundant constraints
can be directly deleted or disabled. Invalidated and conflicting constraints need to be
revised to preventing runtime exceptions. However, in this paper we focus on detecting
process constraint anomalies and leave the anomalies correction to future research.

It is also worth noting that based on the time that process constraints can be evaluated
they can also be classified into two categories: static vs. dynamic. Static constraints can be
checked without running workflow, whereas dynamic constraints can only be enforced and
evaluated during workflow runtime. For instance, constraint c1: t2 (Supervisor Approval)
must be executed by the supervisor of the purchase requestor is a dynamic role resolution
constraint, where the role to execute t2 can only be decided during runtime. Other
well-know dynamic process constraints including separation of duties and binding of
duties. As we will present later, dynamic constraints can transformed into static ones
during runtime for consistency verification. Next, we present the verification rules
defined using our unified predicate language, which enables the automatic detection of
process constraint anomalies.

6.4 Process Constraint Verification

According to the definition of unified predicate language, a process model includes
the following process elements: a set of tasks $T$, a set of routing constructs $RC=\{f, j, b, m\}$,
a set of data items $D$, a set of resources including users $U$, roles $R$, groups $G$, and a set of
constraints $C$. We also define that for any constraint $c$, the involved tasks, routing
constructs, data, resources, and constraints are denoted as $T_c, RC_c, D_c, U_c, R_c, G_c, C_c$.

**Lemma 1 (Condition for Missing Constraint Anomaly).** Missing constraint anomaly occurs if $\exists t_i, \text{count}(\text{execute}(\_, t_i), n), n = 0$ or if $\exists b_j, \text{count}(\text{constraint}(b_j, \_), m), m < 2$.

Discussion: $\text{count}(\text{execute}(\_, t_i), n)$ returns the number of resources assigned to task $t_i$. $n = 0$ means that there is no resource assigned to task $t_i$, therefore a missing constraint anomaly happens. $\text{count}(\text{constraint}(b_j, \_), m)$ returns the number of routing constraints associated with $b_j$. A decision point must have at least outgoing links, which means there must be at least two routing constraints associated with $b_j$. Therefore, when $m$ is less than 2, a missing constraint anomaly occurs.

**Lemma 2 (Condition for Redundant Constraint Anomaly).** Redundant constraint anomaly occurs if the following conditions hold for any constraint $c$: $\forall t_i \in T_c, t_i \notin T$ and $\forall rc_j \in RC_c, rc_j \notin RC$.

Discussion: $\forall t_i \in T_c, t_i \notin T$ means that any tasks defined in the constraint does not exist in the control flow. Similarly, $\forall rc_j \in RC_c, rc_j \notin RC$ means that any routing constructs specified in the constraint does not exist. If the tasks and routing constructs in the constraint do not exist, the constraint will never be evaluated during process execution, which makes it a redundant constraint.

**Lemma 3 (Condition for Invalid Constraint Anomaly).** Invalid constraint anomaly occurs if for any of the following conditions hold for a constraint $c$:

1) $\exists x, x \in U_c \cup R_c \cup G_c, x \notin U \cup R \cup G$

2) $\exists \text{next}(a, b), a \in T \cup RC, b \notin T \cup RC$

3) $\exists e, e \in C_c, e \notin C$. 
Discussion: Conditions 1 states that some resources assigned to a task have been removed resulting in an invalid constraint, because the system could not find the specified resource to execute the task. Condition 2 specifies while a particular task has been removed there are still constraints defining routing path to the deleted task. Condition 3 represents invalid association between non-exist constraints to tasks and routing constructs.

**Lemma 4 (Condition for Conflicting Constraint Anomaly).** Conflicting constraint anomaly occurs if any of the following conditions hold: 1) $\text{execute}(x, y)$ and $\text{cannot_execute}(x, y)$ both exist; 2) $\text{count}(\text{next}(t, _), n)$, $n > 1$

Discussion: Condition 1 means that there exists conflicting resource assignment. Condition 2 means that task $t_i$ has more than one outgoing links.

**Theorem 1 (Process Constraint Consistency).** A set of process constraints is consistent if they are free from missing, redundant, invalid, and conflicting process constraint anomalies, which means the following conditions are satisfied:

1) $\forall t, \text{count}(\text{execute}(\_, t), m) \geq 0$

2) $\forall b, \text{count}(\text{constraint}(b, _), n) \geq 2.$

3) $\forall c, T_c \subseteq T, R_c \subseteq R, U_c \subseteq U, R_c \subseteq R, G_c \subseteq G, C_c \subseteq C$

4) $\exists c_i : \text{execute}(x, y), c_j : \text{cannot_execute}(x, y)$ where $c_i \in C$ and $c_j \in C$

5) $\forall t, \text{count}(\text{next}(t, _), k) \leq 1$

**Proof:** Let Lemmas 1, 2, 3, and 4 be the only situations under which process constraint anomalies can occur. We use enumeration to prove that a set of process constraints will avoid those situations if they satisfy the five conditions in theorem 1. First, $\forall t, \text{count}(\text{execute}(\_, t), m) \geq 0$ ensures that there is at least one resource is assigned to each
task, and $\forall b, \text{count}(\text{constraint}(b_\_), n), n \geq 2$ ensures that each decision point has at least two constraints defining different routing conditions. According to Lemma 1, missing constraint anomaly is avoided. Second, $\forall c, T_c \subseteq T$, $RC_c \subseteq RC$, $U_c \subseteq U$, $R_c \subseteq R$, $G_c \subseteq G$, $C_c \subseteq C$ means that all process elements involved in every constraint exists in the process model. According to Lemma 2 and 3, redundant and invalid constraint anomalies cannot occur. Finally, $\forall c : \text{execute}(x, y)$, $c_j : \text{cannot}_\_\text{execute}(x, y)$ where $c_i \in C$ and $c_j \in C$ prevents contradictory assignment constraints, and $\forall t, \text{count}(\text{next}(t, _\_), k), k = 1$ ensures there is only one outgoing link from each task, which prevents contradictory routing constraints. According to Lemma 4, conflicting constraint anomaly is prevented. Therefore, theorem 1 holds.

Theorem 1 provides the theoretical foundation for automatic detection of process constraint anomalies. Based on theorem 1, we developed a constraint verification algorithm using unified predicate language as shown in Table 31. The algorithm can not only identify four types of process constraint anomalies, it can also pinpoint the process element causing the anomaly, which greatly improves the effectiveness and efficiency of anomaly correction. By applying the algorithm to process constraints shown in Table 29, we can identify four missing policies for tasks $t_1$ (Purchase Request), $t_2$ (Supervisor Approval), $t_4$ (Mgmt. Approval), $t_7$ (Print PO). In particular, $t_1$ can be executed any employee in the organization. We add one meta role ‘Any Employee’ and assign it to $t_1$ by adding a constraint $c_{23}: \text{execute}(\text{’Any Employee’}, t_1)$. As defined in constraints $c_1$ and $c_3$, the roles executing tasks $t_2$ and $t_4$ are dynamically resolved based on role hierarchy. We add
another meta role ‘Dynamic Role’ to execute t_2, t_4 for achieving static process consistency. Therefore, two more constraints are added: c_{24}: execute('Dynamic Role', t_2), c_{25}: execute('Dynamic Role', t_4). In addition, we assign ‘purchasing clerk’ to execute t_7 by adding constraint c_{26}: execute('PC', t_7), which makes process constraints in the procurement approval process consistent.

**INPUT**: Facts on process model including sets of T(tasks), RC(routing constructs), D(data items), U(users), R(roles), G(groups), C(constraints)

**OUTPUT**: 1) SUCCESS if the set of process constraints is consistent
2) MISSING if there is a missing constraint anomaly
3) REDUNDANT if there is a redundant constraint anomaly
4) INVALID if there is a invalid constraint anomaly
5) CONFLICTING if there is a conflicting constraint anomaly

**Procedure** ProcessConstraintVerification

Begin

for each \( t_i \in T \)

\[
\text{count(execute(\_, t_i), m)}; \\
\text{count(next(t_i, \_), n)};
\]

if \( m = 0 \) then

return MISSING and \( t_i \) // missing assignment constraint identified

exit;

end if

if \( n > 1 \) then

return CONFLICTING and \( t_i \) // conflicting routing constraint identified

exit;

end if

findall(execute(\_, t_i), S_1);

findall(cannot_execute(\_, t_i), S_2);

if \( (S_1 \cup S_2) \cap (U \cup R \cup G) = \emptyset \) then

return INVALID and \( t_i \) // invalid assignment constraint identified

exit;

end if

end for

for each \( rc_j \in RC \)

\[
\text{count(constraint(rc_j, \_), n)}
\]

if \( n < 2 \) then

return MISSING and \( rc_j \) // missing routing constraint identified

end if

end for
As we mentioned before, dynamic process constraints can only be evaluated during workflow execution. In order to leverage algorithm in Table 31 for dynamic constraint verification, we adopt generate-and-test technique widely used in logic programming to
transform dynamic constraints into static ones during workflow runtime. Before each task is executed, we first assert new facts as the task has been completed and run the verification algorithm. If anomalies are detected, we suspend workflow execution, retract newly inserted facts, and trigger a dynamic constraint violation alert.

So far, we have verified that process constraints in procurement process are consistent. Next, we propose some process changes in the example processes and show what process constraint anomalies can be detected after those changes, which further validate our constraint verification algorithm.

6.5 Validation of the Constraint Verification Method

In this section, we propose a set of process changes to the procurement approval process and show how the verification algorithm presented in the previous section can help detect different types of process anomalies.

<table>
<thead>
<tr>
<th>Insertion</th>
<th>Deletion</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Control flow</strong></td>
<td><strong>Organization</strong></td>
<td></td>
</tr>
<tr>
<td>$t_{13}$ (Procurement Notification) is added</td>
<td>a new role ‘<em>Vice President (VP)</em>’ is added</td>
<td>$t_5$ (Purchasing) is modified to include inventory checking task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$t_3$ (fund checking), $t_6$ (Print PO), $t_7$ (inventory checking), $t_{11}$ (Archiving), $b_3$ are removed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>previous inventory clerks are reassigned as purchasing clerks</td>
</tr>
</tbody>
</table>

Table 32. Changes in the procurement process

The changes are listed in Table 32. In particular, a new task $t_{13}$ (Procurement Notification) is added to notify the purchase requestor that the item has been shipped or is out of stock. Tasks $t_3$ (fund checking), $t_7$ (inventory checking) are removed to shorten the
approval cycle time and improve overall process efficiency. Due to the integration of ERP and workflow system, tasks $t_6$ (Print PO) and $t_{11}$ (Archiving) are automatically handled by computers in a digital format, which are removed from process model. For organizational model, a new role ‘Vice President’ is added, while role ‘Inventory Clerk’ is removed by assigning all inventory clerks to role ‘purchasing clerk’.

The new control flow diagram is shown in Figure 28 and its corresponding UPL representation is presented in Table 33. The new organization chart is shown in Figure 29, which is represented in UPL in Table 34.

Figure 28. Redesigned procurement process

next($s$, $t_1$), next($t_1$, $t_2$), next($t_2$, $b_1$), branch($b_1$, [$b_2$, $t_{12}$]), branch($b_2$, [$t_4$, $t_5$]), next($t_4$, $b_4$), branch($b_4$, [$t_5$, $t_{12}$]), next($t_5$, $b_5$), branch($b_5$, [$t_8$, $t_9$]), next($t_8$, $b_6$), branch($b_6$, [$t_5$, $m_1$]), next($t_9$, $t_{10}$), merge([$t_{10}$, $b_6$], $m_1$), next($m_1$, $t_{13}$), merge([$t_{13}$, $t_{12}$], $m_2$), next($m_2$, $e$)

Table 33. Redesigned procurement process in UPL
After applying constraint verification algorithm to the process model after changes, we can identify one missing constraint, four redundant constraints, and four invalid constraints. In particular, `count(execute(_, t13), m)` and `m = 0` shows that `t13` (Procurement Notification) has no resource assigned to it. We add a role ‘Agent’ to represent computer system, such as email server, printer, etc. and assign it to `t13` by adding constraint `c27`: `execute('Agent', t13).

For constraint `c2`: `execute(AC, t3)`, `t3` has been removed which makes `c2` never be evaluated. According to Lemma 2, `c2` is identified as a redundant constraint. Similarly, `c5`, `c8`, and `c25` are also identified as redundant. These redundant constraints are directly
removed. Constraint $c_{13}$: $\text{next}(b_2, t_3) \leftarrow \text{greaterEqual}(d_3, 1000)$ defines that $t_3$ is triggered after $b_2$ if $d_3$ is greater or equal to 1000. However, due to the deletion of $t_3$, the evaluation of constraint will lead to a runtime process exception. According to Lemma 3, $c_{13}$ is identified as an invalid constraint. In the same way, $c_{15}$, $c_{16}$, $c_{21}$ are also detected as invalid. According to different constraint type, different actions may be taken to correct invalid process constraint anomalies. For instance, constraint $c_{13}$ should be revised as $\text{next}(b_2, t_4) \leftarrow \text{greaterEqual}(d_3, 1000)$ to reflect the change of task deletion and different routing path. For the binding of duty constraint $c_9$: $t_1$ (Purchase Request) and $t_3$ (Fund Checking) cannot be done by the same person, it can be removed because task $t_3$ has been deleted.

In this section, we use an example to validate our constraint verification algorithm and a set of process constraint anomalies were detected. Next, we discuss the design of a process change analysis tool developed based on the constraint verification algorithm.

6.6 Design of a Process Change Analysis Tool

Once a process model is formalized using process designer, a corresponding process representation in our Unified Predicate Language (UPL) can be generated. Then, the algorithms for design correctness checking and constraint verification presented in section 5.2 and 6.4 can be applied. We have been developing a logic-based process change analysis tool named Process Change Analyzer (PCA) by adopting the well-know Java-based rule engine (http://www.jessrules.com/) and map our UPL into Jess language.

Jess is one of the most popular Java-based rule engines developed at Sandia National Laboratories (Friedman-Hill 2003). Jess is free for academic use and has many
unique features including backwards chaining and working memory queries, and
capability of directly manipulating and reasoning about Java objects. Jess also provides a
powerful Java scripting environment, where Java objects can be created and Java APIs
can be called. Jess program can also be invoked in Java program, which enables easy
integration between Jess-based Process Change Analyzer with Java-based process design
tools we discussed in section 4.5. In addition, Jess also provides an IDE based on Eclipse,
which further simplify Jess application development as shown in Figure 30.

Figure 30. Constraint verification algorithm implemented using Jess

Given that the syntax of UPL is similar to Jess language, the mapping between UPL
predicates to Jess constructs is easily implemented. In particular, fact templates are used
to define the routing constructs fork, join, branch, and merge, which include unordered
list of multiple tasks. Sequential execution, data flow, and organizational models are
simply mapped into ordered fact lists. Then, the algorithm for constraint verification is
translated into a set of Jess rules. Another set of Jess functions are created to control Jess
program logic and the interaction with users. Process changes are carried out by
modifying the fact base using Jess build-in function ‘assert’ and ‘retract’. ‘assert’ inserts
a new fact into the fact base, whereas ‘retract’ removes a fact, e.g., assert (next t2 t5)
inserts a new link between task t2 and t5. Whenever the fact base is changed, i.e., the
process model is changed, the verification rules are triggered and the verification results
are inferred.
7. CONCLUSIONS

Today, workflow management systems have been widely deployed in organizations to help streamline their business processes, improve product quality, and enhance customer satisfaction. Process analysis and design are the most important phase in workflow application development, determine the overall structure and quality of the workflow model, and therefore largely decide the result of process automation: success or failure. One key requirement of process analysis and design is to make sure that resulting workflow models are compliant with organization policies. Due to new government regulations such as Sarbanes-Oxley and quality assurance standards such as ISO9001, many organizations are revising their policies and business processes in order to achieve compliance. Given those frequent changes, maintaining consistency between business policies and processes can be a very challenging task. Therefore, there is a need for advanced process analysis and design methods to better support dynamic policy and process changes.

In this dissertation, we provide solutions by proposing a logic-based methodology for process analysis and design, which builds a formal linkage between business policies and workflow models. Our contribution is threefold. First, we advocate a new Policy-driven Process Design (PPD) methodology which is different from existing participative and analytical methods. PPD is more efficient, effective, and accurate than participative approaches. It avoids traditional time-consuming, ambiguous process data collections by leveraging and systematically analyzing detailed process policies. It is also more practical
and intuitive than analytical approaches, because it offers specific process design rules and detailed procedure for extracting workflow models from narrative business policies. Several steps in that procedure can be automated and a set of algorithms have been developed for that purpose. As far as we know, PPD is the first attempt towards policy-based process design automation. Furthermore, PPD provides a unique policy perspective on process analysis and design, where different workflow components are unified as policies.

Second, we propose a predicate logic-based process modeling language to formally represent the unified policy view of workflow models, which we refer to as Unified Predicate Language (UPL). In UPL different workflow components, including control flow, data flow, organizational model, and process constraints, are represented as logic formulas. This unified logic format enables analysis of the interaction among different workflow components via logic inference and queries. In particular, we demonstrate the expressive power and analytical capability of UPL by applying it in process design. UPL consists of a succinct set of predicates that are also expressive enough to model most commonly used control flow structures, data dependencies, organizational hierarchy, and process constraints. It also provides a set of functions that can be used to develop algorithms for offering design insights and detecting design problems. UPL is compared to other process modeling languages and the result shows that UPL is the only analytical process modeling language that can model all workflow components and support intelligent process design.

Third, while most existing research on process changes only focus on one workflow
aspect, generally control flow, we emphasize the analysis of changes between different workflow components. In particular, we thoroughly investigate how changes in control flow and organizational model can affect process constraints. We summarize all possible scenarios where process constraints can be compromised due to process changes and formally define those scenarios into four process constraint anomalies using our proposed Unified Predicate Language, namely, missing constraint, redundant constraint, invalid constraint, and conflicting constraint. We demonstrate that process constraint anomalies can lead to serious runtime workflow errors. We prove that if none of these four process constraint anomalies exist the process constraints are consistent. A verification algorithm is developed to verify process constraint consistency whenever process changes happen and a constraint enforcement algorithm is designed to evaluate dynamic process constraints during runtime.

In summary, this dissertation fills a critical void in business process management by bridging the gap between commercial needs for advanced process analysis and design methodology and existing workflow research. As such, our research will have important impact in both theory and practice.

We are planning to extend our research in a number of directions. First, we are exploring methods for automatic process policy extraction from narrative business policies. In particular, we are investigating the possibility of leveraging semantic web research on rule markup languages, such as XRML (Lee et al. 2003) and RuleML (Grosof 2001). Similar to the procedure described in (Lee et al. 2003), narrative business policies are first analyzed by a knowledge engineer, where process related information is marked
using predefined XML tags. Then, process information can be automatically extracted by applying information retrieval techniques.

Second, we are interested in analyzing semantic errors in process design. Compared to syntactical errors that can be detected by formal methods, semantic errors can only be identified by human inspection, which is error-prone and inefficient. We plan to first work on summarizing and classifying process design semantic errors. Then, based on the different types of semantic errors, we will try to provide systematic procedures to help prevent and detect semantic errors.

Third, we continue developing and enhancing our prototype system for policy-driven process design and logic-based process change analysis, which includes Policy Wizard (PW), Visual Editor (VE), and Process Change Analyzer (PCA). Once the system development is complete, we will test the usability of our system by conducting user studies and further validate our methodology by doing controlled experiments. In particular, we want to demonstrate that PW and VE can significantly improve the efficiency of process design and PCA can perform the task of identifying process change anomalies better or as well as human process analysts. Many techniques on information system verification and validation can be leveraged to design our user studies and experiments (Agarwal et al. 1993a; Sakthivel et al. 1988). The experiment results can help us refine our logic-based process analysis and design methodology and enhance our prototype system.
REFERENCES


Bajaj, A., and Ram, S. "SEAM: A state-entity-activity-model for a well-defined workflow development methodology," *Ieee Transactions on Knowledge and Data


Sun, S.X., and Zhao, J.L. "A Data Flow Approach to Workflow Design," Proceedings of


