

Industry Foundation Processes (IFP):
Theoretical and Practical Foundations
for the Construction Industry

by

Behrooz Golzarpoor

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Civil Engineering

Waterloo, Ontario, Canada, 2017

©Behrooz Golzarpoor 2017

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

ABSTRACT

Industry foundation processes are formulated to improve capital project process conformance and interoperability. These processes are used to implement key elements of practices. Several research studies confirm that the implementation of best practices drives better engineering and construction project performance. Best practices are defined by the Construction Industry Institute (CII) as processes or methods that when executed effectively, lead to enhanced project performance. Particular organizations, such as the CII, the Construction Owners Association of Alberta (COAA), and the Project Management Institute (PMI), develop and promote best practices pertaining to various aspects of capital project delivery. However, the systematic and consistent implementation of such practices throughout the lifecycle of a construction project and from project to project remains a challenge.

Research findings also reveal that improved adoption of best practices, through conformance with their processes, and improved interoperability, are correlated with substantial capital project performance improvements in terms of cost, schedule, and productivity. In many industry sectors, such as health care, manufacturing, and banking, process conformance has been radically improved through the automation of processes via workflow engines, and several efforts are being made to regulate standards to facilitate process interoperability. However, process conformance and interoperability in the construction industry are lagging behind. In the construction industry, a promising solution for facilitating effective and consistent conformance with best practices lies in the employment of workflow processes and workflow engines.

The concept of Industry Foundation Processes (IFP) and the theory and framework for IFP development and implementation are established in this research. The objective is to integrate construction industry best practices into Electronic Product and Process Management (EPPM) systems, and improve process interoperability and conformance. EPPM systems, which are increasingly being used for managing mega capital projects, can be described as the meta-managers of other systems, such as document management systems (DMS), building information modeling (BIM), workflow management systems (WfMS), and advanced project management systems. Integration of best practices into EPPM systems facilitates more consistent and scalable

adoption of best practices in large-scale construction projects, resulting improved project performance.

IFPs are defined as standard workflows based on known best practices in the construction industry with certain features and characteristics to improve process conformance and facilitate process interoperability. The research methodology is comprised of four main phases: (1) developing methods and mechanisms that can be used to transform best practices into structured workflow process in such a way as to retain the essence of the best practices, (2) defining the IFP concept and establishing a framework and an ontology for inheritance and customization of IFPs for specific corporate and project circumstances, (3) customizing and implementing particular IFPs in an EPPM system, based on available records for specific construction projects, and investigating the applicability and effectiveness of the IFP concept, and (4) analyzing and validating the value of the IFP system through functional demonstration of the benefits, including process conformance and interoperability.

The scope of the thesis is the theoretical development of IFP system, in addition to implementation studies for a limited number of IFP processes within the domain of industrial sector construction projects. The development and application of the IFP system is anticipated to result in more effective adoption of best practices and enhanced process conformance and interoperability, with the end-result of improved capital project performance.

ACKNOWLEDGEMENTS

First and foremost, I am extremely grateful to my advisor, Professor Carl T. Haas, for his valuable guidance, encouragement, and support during my PhD program. He is an exceptional supervisor and a wonderful mentor, from whom I have learned invaluable lessons – directly and indirectly. I have no doubt that his impact in my life transcends the duration of my PhD program.

I was also privileged to have the support, constructive feedback, and helpful advice of Professor Keith Hipel, Professor Frank Safayeni, and Professor Tarek Hegazi as members of my PhD examination committee. Their insightful comments significantly improved my thesis. I would also like to thank Professor Vineet Kamat, the external committee member of my PhD defense, for his valuable comments and suggestions.

This research was not possible without the interdisciplinary collaboration and contribution of several people. I am sincerely grateful to Professor Derek Rayside, from the Electrical and Computer Engineering department, for the effective collaboration we had, and for his indispensable expertise required in this research. I would also like to express my gratitude to the co-op students Matthew Weston, Tao Lue, and Ming Zhou for their contributions in the progress of this research study, and particularly for their programming skills.

The support of Coreworx Inc. is truly appreciated. I am grateful to Ray Simonson, Peter Walker, Paul Harapiak, Kelly Maloney, and particularly Joel Gray for providing me technical resources, expert feedback, and helpful advice. I would also like to acknowledge the financial support of Natural Sciences and Engineering Research Council of Canada, Coreworx Inc., and the Ontario Graduate Scholarship.

Last, but not least, my special thanks go to all of my family members, and especially to my loving wife Samaneh Shadmehr, for her unconditional love and continued support.

DEDICATION

*To my parents,
and my wife, Samaneh*

TABLE OF CONTENTS

Declaration	ii
Abstract	iii
Acknowledgements	v
Dedication	vi
Table of Contents	vii
List of Figures	x
List of Tables	xii
Chapter 1 Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Problem Statement and Research Need	4
1.4 Research Objectives	6
1.5 Research Scope	7
1.6 Research Methodology	7
1.7 Thesis Structure	9
Chapter 2 Literature Review	11
2.1 Introduction	11
2.2 Construction Industry Best Practices	12
2.2.1 Best Practices as a Form of Knowledge	14
2.3 Process Management	15
2.3.1 Process – Definition and Levels	16
2.3.2 Process Modeling	17
2.3.3 Process Modeling Tools and Standards	19
2.3.4 Process Specialization	20
2.3.5 Process vs. Practice	21
2.3.6 Workflow vs. Process	22
2.4 Information Management Systems	23
2.4.1 Workflow Management Systems (WfMS)	23
2.4.2 EPPM System	24

2.5 Interoperability – Definition and Levels	25
2.6 Interoperability in AEC/FM Domain	27
2.7 The Knowledge Gap	28
Chapter 3 Industry Foundation Processes (IFP)	29
3.1 IFP Modeling System and IFP Processes	29
3.2 Approaches of Developing IFP Processes	30
3.3 Extracting the Common Core of Implemented Processes	31
3.4 Defining IFPs Based on Well-Known Best Practices	38
3.4.1 Abstract Framework	39
3.4.2 Pragmatic Framework	40
3.5 Discussion	44
Chapter 4 Proposed IFP Ontology	46
4.1 IFP Ontology	46
4.2 Version and Scope	47
4.3 Core Structure	47
4.4 Abstraction Level	49
4.5 Data Structures	50
4.6 Recommended Practice	51
4.7 Inheritance	51
4.8 Conformance	54
4.9 Interoperability	56
Chapter 5 IFP System Validation	57
5.1 Validation Methodologies	57
5.2 IFP System Deployment	58
5.2.1 Deciding on the Deployment Platform	59
5.2.2 Workflow Foundation (WF) Technology	60
5.2.3 Implementation of RFI Workflow Process	61
5.3 Discrete Event Simulation (DES)	63
5.3.1 Simulation of RFI Workflow Process	64
5.3.2 Modeling and Simulation using SIMUL8	67
5.3.3 Simulation Analysis	69

5.4 Discussion	70
Chapter 6 Improving Process Conformance with IFP	72
6.1 Process Conformance.....	72
6.2 Workflow Conformance Checking	73
6.3 Conformance Checking Algorithm	73
6.4 The Alloy Language and Its Advantages	76
6.5 Workflow Conformance Checking using Alloy	78
6.6 Validation Case Study.....	81
6.7 Automated Workflow Conformance Checking Tool.....	83
Chapter 7 Improving Process Interoperability with IFP	85
7.1 Process Interoperability.....	85
7.2 Process Interoperability Approaches	86
7.3 Process Interoperability in AEC/FM Domain.....	87
7.4 IFP Interoperability Model.....	88
7.5 Implementation Using Workflow Foundation (WF) Technology	91
7.6 Discussion	95
Chapter 8 Conclusions and Future work.....	97
8.1 Summary and Conclusions.....	97
8.2 Contributions.....	98
8.3 Limitations	99
8.4 Recommendations for Future Work.....	100
References.....	102
APPENDIX A Glossary of Terms	112
APPENDIX B Samples of Core WF Code for Deployment of RFI Process.....	114
APPENDIX C Work Completed Under My Supervision to Support Validation of Conformance Checking (Tao Lue Wu, 2015)	130
APPENDIX D Alloy Code for RFI Workflow Conformance Checking.....	147
APPENDIX E Translator.java Documentation	151
APPENDIX F Translator.java Code.....	154
APPENDIX G Automator.java Documentation	166
APPENDIX H Automator.java Code	167

LIST OF FIGURES

Figure 1-1: Mechanical Construction Productivity vs. High and Low Level of Best Practices Implementation (Shan et al., 2011).....	3
Figure 1-2: Productivity Comparison by Trades for High and Low Levels of Construction IT Integration (Zhai et al., 2009)	3
Figure 1-3: Various Approached for Adoption of Best Practices	4
Figure 1-4: IFP Research Rational.....	6
Figure 1-5: Research Objectives	7
Figure 1-6: Research Methodology	8
Figure 2-1: Knowledge Hierarchy and Its Association with Tacit & Explicit Knowledge, and with Practice and Process.....	15
Figure 2-2: Classification of Business Processes (Weske, 2012)	17
Figure 2-3: Typical Users and Tools for Each Process Level	18
Figure 2-4: Three Levels of Interoperability (Lewis, 2013)	25
Figure 3-1: Approaches of Developing IFP Processes	31
Figure 3-2: Different Implemented Versions of the RFI Process in Skelta Software Format.....	33
Figure 3-3 (a): High-Level Representation of the RFI Workflow Process in Project A	34
Figure 3-3 (b): RFI High-Level Representation of the RFI Workflow Process in Project B	35
Figure 3-3 (c): High-Level Representation of the RFI Workflow Process in Project C.....	36
Figure 3-4: The Common Core Structure of the RFI Workflow Process	37
Figure 3-5: Transforming a Practice into a Structured Process	41
Figure 3-6: CII Change Management Principles, Each Offered as an Organizational Process.....	42
Figure 3-7: Main Steps of a Change Request (CR) Workflow Process.....	43
Figure 3-8: A Change Request (CR) Process in BPMN Notation	44
Figure 4-1: Proposed IFP Ontology	46
Figure 4-2: The Core Structure of an IFP for the RFI Process	48
Figure 4-3: Examples of Accepted and Prohibited Transformations.....	54
Figure 4-4: Conformant and Non-Conformant Versions of the RFI Process	55
Figure 5-1: Example of Programming Inheritance for Respond Activity	59
Figure 5-2: Implementation of the RFI-IFP Workflow as a State Machine Model.....	61
Figure 5-3: The Coordinator View	62

Figure 5-4: The Consolidator View	63
Figure 5-5: An RFI Workflow Process Used in a Capital Mega Project.....	65
Figure 5-6: A Snapshot of the Simulation Model in SIMUL8	69
Figure 6-1: Examples of Directed Graphs That Are Not Well-Formed	74
Figure 6-2: Alloy Implementation of Workflow Process	78
Figure 6-3: Alloy Implementation of Well-Formed Workflow Process.....	79
Figure 6-4: Alloy Specification (Excerpt) of Workflow Conformance for Steps 4 and 5 of the Algorithm.....	79
Figure 6-5: Visualization of Conformance Checking for Workflow W9 of Figure 4-3	80
Figure 6-6: Conformance Checking Analysis of a Non-Conformance RFI Workflow Process.....	82
Figure 7-1: IFP Interoperability Model for Interaction of Two RFI Workflow Processes.....	89
Figure 7-2: IFP Interoperability Model for Interaction of a CR and an RFI	90
Figure 7-3: Modeling RFI Customized Workflows.....	92
Figure 7-4: An Overall Exchange Record	93
Figure 7-5: Examples of Data Objects of an Exchange Record	94
Figure 7-6: A Snapshot of Message Exchange Between Activities of Two RFI Processes.....	95

LIST OF TABLES

Table 2-1: Construction Industry Best Practices	12
Table 2-2: CII Best Practices	13
Table 2-3: COAA Best Practices	13
Table 2-4: Summary of Knowledge Hierarchy.....	14
Table 2-5: Process Modeling Tools	19
Table 2-6: Process vs. Practice	22
Table 2-7: Process vs. Workflow.....	23
Table 2-8: Four Levels of Interoperability.....	26
Table 3-1: Types of Knowledge in a Practice and their Association with Process Elements	40
Table 3-2: Evaluate Change Process.....	42
Table 4-1: Workflow Abstraction Levels	49
Table 4-2: Minimal Set of Data Structure Fields for an RFI Process.....	51
Table 4-3: Sample of a RACI Chart	52
Table 4-4: Sample of Workflow Inheritance Rules	53
Table 5-1: Versions of the RFI Workflow	64
Table 5-2: Data Fields and Their Description.....	66
Table 5-3: A Sample of Retrieved RFI Workflow Process Enactment Data.....	66
Table 7-1: A Sample of Common Workflow Processes in Large Construction Projects.....	87

Chapter 1

Introduction

1.1 Background

Construction of large-scale capital projects are huge undertakings with inherent complexities. Large numbers of project stakeholders, overlap of construction activities, variety of technologies employed, several trades that are involved, and the uncertainty and risk in the design, procurement, and construction of such projects, create technical, organizational, and social complexities. Severe competition and increased demand for faster delivery, while maintaining high quality engineering standards, further add to these complexities.

Traditional project management controls that are based on linear critical path method (CPM) schedules and earned value analysis are no longer adequate for successful delivery of such projects. To deal with such complexities, more dynamic Integrated Project Delivery (IPD) approaches that employ technologies such as Interface Management (IM) and Building Information Modeling (BIM) are required to integrate people, systems, business structures, and practices via employment of workflow engines and workflow processes. These more recent approaches rely on highly effective coordination and timely communication among many project stakeholders, real-time tracking and measurement of the project's progress and performance, early detection of risk, and minimizing but rapidly adapting to imperative change.

Consequently, over the years, supporting information systems evolved from conventional data-aware systems to modern process-aware systems. Data-aware information systems evolved around centralized database management systems (Weske, 2012). Today's process-aware information systems facilitate interaction and collaboration of stakeholders via distributed systems (Wil M. P. van der Aalst, 2014). Examples include advanced project management collaboration tools, enterprise resource planning (ERP) systems (Chung, Skibniewski, & Kwak, 2009; Ghosh, Negahban, Kwak, & Skibniewski, 2011; O'Connor & Dodd, 2000; Skibniewski & Ghosh, 2009), workflow engines (Wil M. P. van der Aalst, 2004; Cardoso, Bostrom, & Sheth, 2004; Tang & Akinci, 2012), electronic document management systems (Al Qady & Kandil, 2013; Caldas, Soibelman, & Gasser, 2005), knowledge-based information systems (El-Gohary & El-Diraby,

2010; Youngcheol Kang, O'Brien, & O'Connor, 2012), and more specifically electronic product and process management (EPPM) systems (Shahi, Haas, West, & Akinci, 2014; Shokri et al., 2012).

EPPM systems, which are increasingly being used in managing mega capital projects (Shahi et al., 2014), are most simply characterized as meta-managers of other systems. They are process-based and workflow-driven. They provide interfaces with building information modeling, enterprise resource planning, and advanced project management systems for information exchange and interoperability among those systems throughout the project lifecycle. Their core components include a document management system, a collaboration management system, and a workflow management system to support various construction workflow processes, such as change management, procurement management, request for information, contract management, and interface management. As a result of these unique characteristics, EPPM systems are the right platform and technology to facilitate consistent integration of construction industry processes and practices throughout the lifecycle of a construction project and from project to project, with the end result of improved project performance.

1.2 Motivation

Several research studies (El-Mashaleh, O'Brien, & Minchin, 2006; Y. Kang et al., 2013; Y. Kang, O'Brien, Thomas, & Chapman, 2008; S. Lee et al., 2005; Shan, Goodrum, Zhai, Haas, & Caldas, 2011; Thomas, Lee, Spencer, Tucker, & Chapman, 2004; Zhai, Goodrum, Haas, & Caldas, 2009) confirm that identification and adoption of best practices and integration of information technologies (IT) drive performance and productivity improvement. For example, Figure 1-1 and Figure 1-2 demonstrate productivity comparison in projects with high and low levels of best practice implementation, and in projects with high and low levels of IT integration, respectively. Moreover, research studies emphasize that although productivity improvement in engineering and construction can be pursued in a variety of ways, gaining faster and more sensible results is probable through increased adoption of best practices in management of projects (Chanmeka, Thomas, Caldas, & Mulva, 2012).

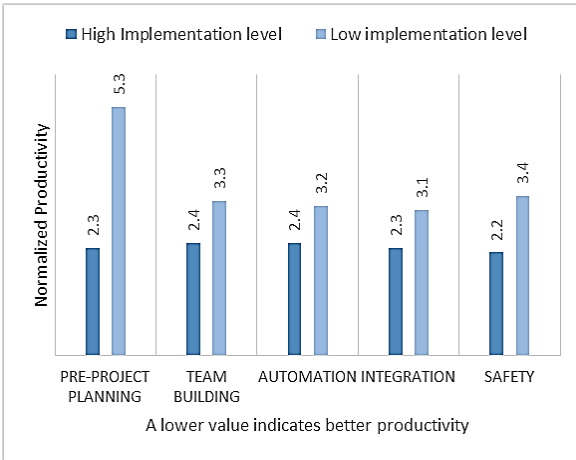


Figure 1-1: Mechanical Construction Productivity vs. High and Low Level of Best Practices Implementation (Shan et al., 2011)

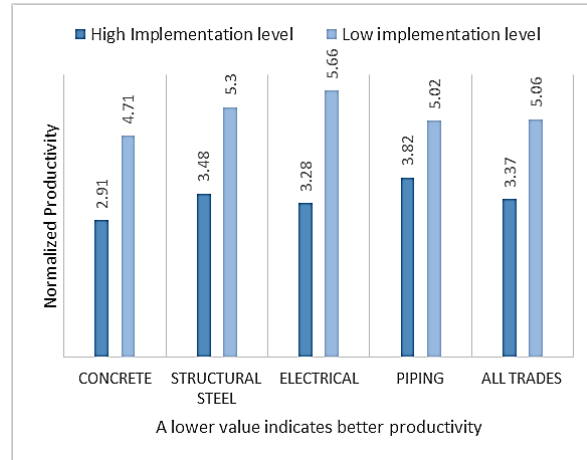


Figure 1-2: Productivity Comparison by Trades for High and Low Levels of Construction IT Integration (Zhai et al., 2009)

Identifying the value of best practices in project performance, well-known organizations, such as the Construction Industry Institute (CII), the Construction Owners Association of Alberta (COAA), and the Project Management Institute (PMI), are developing and promoting best practices in connection with various aspects of capital project management and delivery. According to CII, best practices are processes or methods that provide improved results when implemented effectively, and thus, can lead to enhanced project performance.

However, the systematic and consistent implementation of such practices throughout the lifecycle of construction projects and from project to project remain a significant challenge. Traditional approaches of adopting best practices include socialization and face-to-face interactions, such as meetings, workshops, and training, which are not easily scalable for implementation of best practices in large-scale capital projects. An alternative solution is to transform best practices into workflow processes and utilize business process models and workflow engines to facilitate effective and consistent conformance to best practices. Figure 1-3 illustrates this viewpoint.

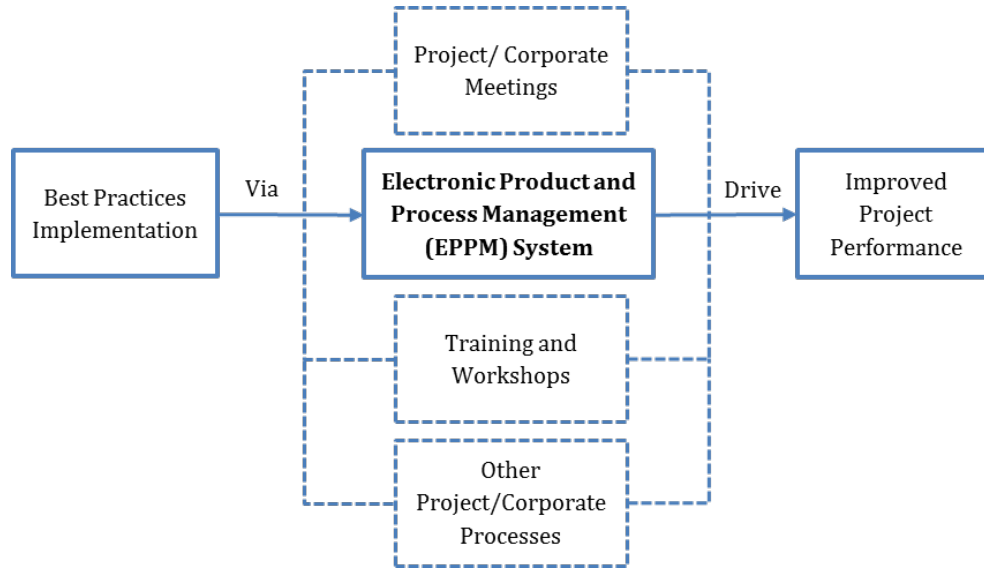


Figure 1-3: Various Approached for Adoption of Best Practices

Increased use of process-based and workflow-driven systems, such as EPPM systems in managing mega capital projects and fundamental improvements in communication and collaboration technologies provide the required resources and the right infrastructure, to facilitate putting this approach into practice. This is the motivation for this research, to facilitate integration of best practices into EPPM systems, to enhance process conformance and interoperability, with the ultimate objective of improving capital projects performance. Employment of workflow engines and EPPM systems to facilitate conformance with best practices offers the advantages of consistency, accuracy, and scalability, and can be considered a key methodology for adopting best practices in mega capital projects.

1.3 Problem Statement and Research Need

Process conformance and interoperability are long sought after goals in capital facility engineering and construction project management. Processes are defined within corporate operating standards by the most sophisticated firms, but study after study confirms that they are not implemented consistently from project to project (Chanmeka et al., 2012; Y. Kang et al., 2008). Process conformance in many industry sectors such as health care, manufacturing, and banking has been radically improved with automation and integration of processes via workflow engines.

While process automation through workflows promises to help substantially improve process conformance, and thus capital project performance, it is being done to date in an ad hoc manner

that is neither scalable nor easily and systematically adaptable to different organization and project circumstances.

For instance, change management process in each organization is typically defined based on the unique needs and existing settings of that organization, resulting very different implementations in each organization. A process may even be implemented differently from project to project within the same organization. Consequently, it is not unusual in large-scale capital projects that a number of collaborating firms possess very different implementations of the same process – e.g., change management or risk management – implemented into their legacy systems. Since these unique implementations do not comply with a common foundation for implementation of that process, process conformance and interoperability among these systems would not be achievable.

The current approach to deal with this problem is to ignore each firm's legacy systems and their processes, and enforce the use of one software platform by all the firms involved in the project. This enforcement is typically performed by the owner or the main EPC contractor, either by imposing the use of a particular software platform through contract terms and conditions, or by providing a cloud based software platform to be used by all parties involved in the project. This approach, however, negatively effects the total time and cost of the project due to the extra training required for employees who must use a new and unfamiliar software platform in each project.

The existing approach disregards the need for interoperability among existing systems and offers a completely new system to be substituted for the legacy ones. A study by the U.S. National Institute of Standards and Technology (NIST) in 2004 estimates the cost of inadequate interoperability among computer-aided design (CAD), collaboration and information systems, and other software systems in the American capital facilities industry to be more than \$15 billion per year (GCR, 2004).

What is needed is a standard implementation of common processes based on industry best practices. Incorporation of processes that comply with a common core into EPPM systems, facilitates process conformance, and supports process interoperability among different systems used by all parties involved, within different phases of a project and among multiple projects.

1.4 Research Objectives

This research and its objectives are based on the following three key premises summarized in Figure 1-4:

1. Most known construction industry best practices are process-based or can be defined as processes, and thus, can be the basis for developing Industry Foundation Processes (IFP).
2. IFP templates can be defined in such a way as to be customizable, and customized versions of IFPs can be rigorously and methodically derived from the Foundation Processes for specific project conditions, similar to IFC implementation.
3. IFP implementation through workflow management systems not only promotes conformance to best practices throughout the project life-cycle, but also offers improved interoperability within project phases and among different projects.

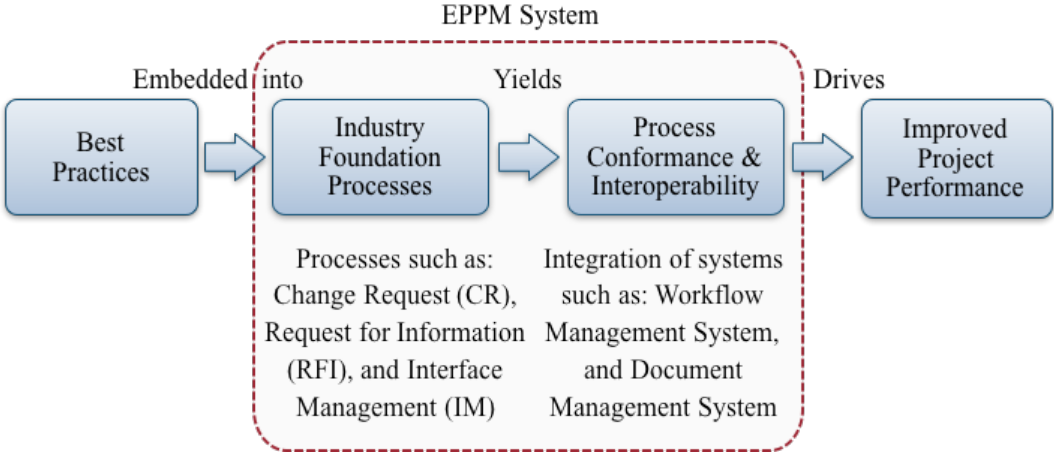


Figure 1-4: IFP Research Rational

Aligned with the premises, the objectives of the research are: (1) to develop a novel theory and process modeling system, called Industry Foundation Processes (IFP), (2) to establish a framework for their application and implementation in such a way as to facilitate integration of core processes of known best practices in the construction industry into workflow management systems, and (3) to improve inter- and intra-projects’ process conformance and interoperability. The ultimate result should be capital project performance improvements. These objectives are illustrated in Figure 1-5.

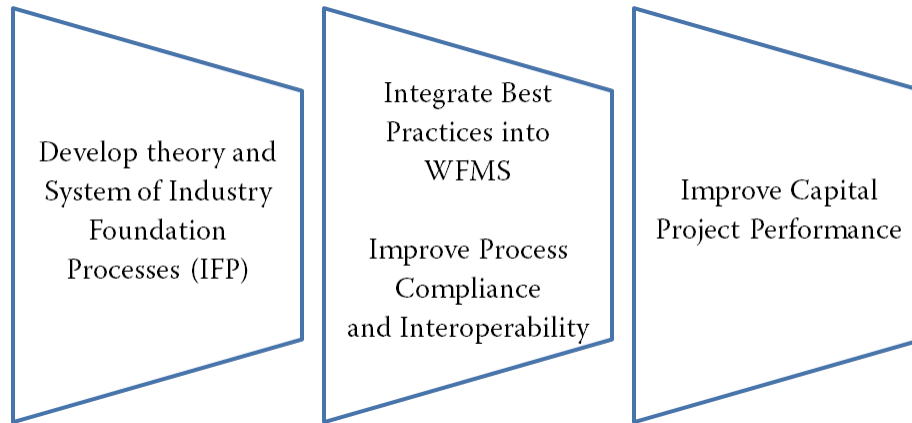


Figure 1-5: Research Objectives

IFPs¹ are defined as workflow templates that can be customized for specific projects’ circumstances and conditions. A workflow engine is used to manage and execute processes enclosed in workflows, and an EPPM system manages the interactions within the whole system. The EPPM system not only supports best practices conformance and interoperability through IFP model implementation, but it also provides automation and integration of other systems and services, thus, facilitating improved project performance.

1.5 Research Scope

This research concentrates on the mechanisms and methods of developing Industry Foundation Processes and establishing a framework and ontology for IFP theory and application. The scope of this research, thus, is essentially the theory development for industry foundation processes, in addition to the implementation of a limited number of IFPs for the domain of industrial sector construction projects. Development and implementation studies of IFPs for several other known best practices in the construction industry, as well as the application of the system of IFPs to other sectors can be addressed in other future research initiatives.

1.6 Research Methodology

This research started with a comprehensive literature review including workflow management systems, construction industry best practices, conformance and interoperability, data and process

¹ In this research, the “IFP” acronym for Industry Foundation Processes is used to refer to the IFP modeling system as well as to a single IFP process. The plural form “IFPs” refers to more than one IFP process.

modeling standards, and process modeling and simulation tools. This review resulted in a more-precise definition of methodology and identification of required tools and techniques, required for performing the next research steps.

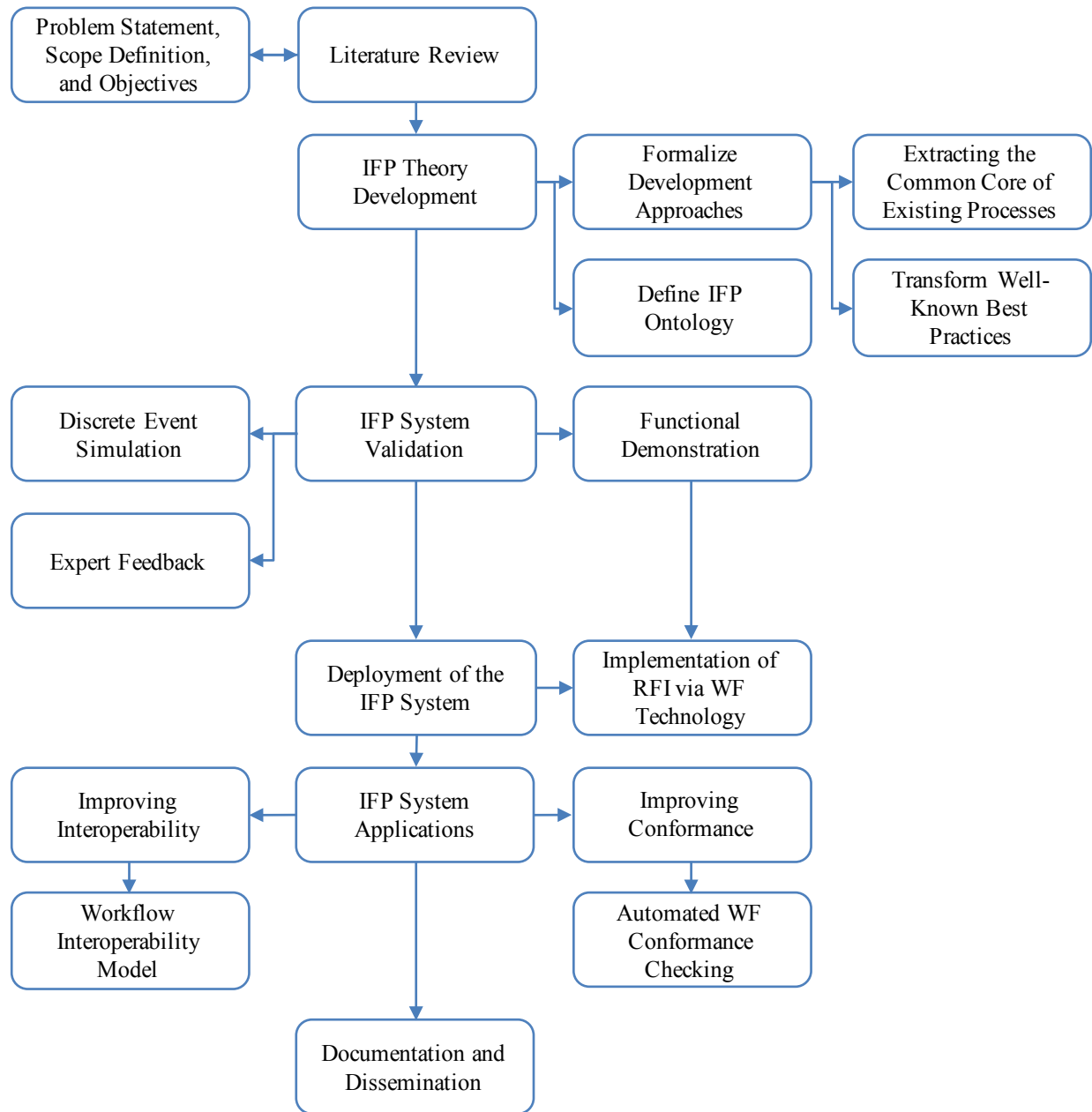


Figure 1-6: Research Methodology

Following and based on the literature review, this research was comprised of six distinct phases: (1) defining the theory and introducing the concept of Industry Foundation Processes (IFP); (2) developing a framework for transformation of industry best practices into structured

processes; (3) establishing an ontology for the IFP system and defining the required components; (4) validating the functionality of the IFP system by implementing a sample IFP process in a workflow management system; (5) developing of an automated conformance checking tool using a first-order-logic programming language to compare workflow processes and check the conformance of a customized workflow process with an IFP; and (6) developing a process interoperability model based on the IFP system to facilitate interoperability of IFP conformance workflow processes. The steps of the research methodology are presented in Figure 1-6.

1.7 Thesis Structure

This thesis is organized in eight chapters. An overview of the research, which includes research need and motivation, hypothesis and objectives, scope, and methodology, is provided in Chapter 1. Chapter 2 provides the literature analysis and the background on several relevant topics such as construction industry best practices, process management, process modeling, workflow management systems, EPPM systems, process conformance, process interoperability, and the gaps and limitations of current studies. Chapter 3 introduces the concepts of foundation-level processes and industry foundation processes (IFP). This chapter defines the features and characteristics of IFP and offers two approaches for IFP development. These approaches are discussed in more detail with prototype examples of common processes used in large-scale capital projects.

A framework and ontology for IFP system is proposed in Chapter 4. The proposed ontology includes eight components and provides the basis for IFP workflow inheritance. It introduces workflow customization mechanisms and conformance metrics for IFP processes.

Validation approaches for the IFP system are discussed in Chapter 5, including expert feedback, discrete event simulation, and functional demonstration. Deployment of the IFP system by implementation of request for information (RFI) workflow via Microsoft Windows Workflow Foundation (WF) technology is presented in this chapter as part of the functional demonstration validation methodology.

Chapters 6 and 7 explore applications of the IFP system. In Chapter 6, a first-order-logic programming language is used to develop an algorithm for comparing the structure of two workflow process. Employing this algorithm, an automated workflow conformance checking tools

is developed by which the conformance of any workflow process with an IFP process can automatically be analyzed and visualized. Chapter 8 proposes an interoperability model to facilitate exchange of information between workflow processes that conform to the IFP system. Finally, the conclusions and future work is the subject of Chapter 8.

Chapter 2

Literature Review

2.1 Introduction

Traditionally, information systems have played a vital role in managing a business, enterprise, or project by supporting improved decision making. They have been widely used for creating, organizing, storing, retrieving, manipulating, and distributing information, and have had a positive impact on productivity and performance. Over the years, however, their applications and scope have been expanded from conventional data-aware information systems, such as database management systems, to process-aware information systems, such as business process management (BPM) and workflow management systems (WfMS) (Wil M. P. van der Aalst, 2014; Weske, 2012). Conventional data-centric information systems are still an important backbone of modern information systems (Weske, 2012), but today's information systems rely on efficient and effective processes and best practices.

In the domain of the construction industry, several research studies (El-Mashaleh et al., 2006; Y. Kang et al., 2013, 2008; S. Lee et al., 2005; Shan et al., 2011; Thomas et al., 2004; Zhai et al., 2009) have confirmed that adoption of best practices and utilization of information technology (IT) and more specifically project management information systems (PMIS) drive substantial performance and productivity improvements. The more recent findings (Y. Kang et al., 2013; Youngcheol Kang, O'Brien, & Mulva, 2013), however, revealed that improvements of automated work processes via information systems is in fact the main driver of improved project performance, and thus signified the importance of well-defined processes and best practices. Based on a statistical analysis of 133 construction projects from the Construction Industry Institute Benchmarking and Metrics database, they concluded that using information systems without enough attention to practices has a limited benefit for project performance, but the combined adoption of best practices and employment of information systems has a more significant impact on project performance. Their study challenged the common belief of strong direct correlation between employment of information systems and improved project performance, and suggested

shifting focus to improvement of work processes to be more efficient or effective by adoption of best practices.

This chapter is a synthesis of a literature review of construction industry best practices, process management, and information systems to form the background for this research.

2.2 Construction Industry Best Practices

It is well established from statistical analysis of hundreds of projects that effective implementation of best practices is correlated with substantial improvements in project performance in terms of cost, schedule, and productivity. Research studies state that systematic implementation of best practices is one of the most important contributing factors to mega projects' success (Chanmeka et al., 2012). A best practice might be a single procedure or method, but most usually it is a combination of several policies, rules, procedures, and methods, in a particular domain.

Several organizations, such as the Construction Industry Institute (CII), the Construction Owners Association of Alberta (COAA), and the Project Management Institute (PMI), develop and promote industry best practices relevant to different aspects of capital project delivery. Best practices are also identified with some other terms, such as Value Improving Practices, Professional Practices, Recommended Practices, and Standards of Practice. Table 2-1 presents a list of organizations that develop and promote such practices and includes their associated terms.

Table 2-1: Construction Industry Best Practices

ORGANIZATION	GUIDELINES REFERRED AS
Construction Industry Institute (CII)	Best Practices
Construction Owners Association of Alberta (COAA)	Best Practices
Independent Project Analysis (IPA)	Value Improving Practices (VIPs)
Project Management Institute (PMI)	Foundational and Practice Standards
Construction Management Association of America (CMAA)	Standards of Practice
The Association for the Advancement of Cost Engineering (AACE) International	Professional Practice Guides (PPGs)
The American Institute of Architects (AIA)	AIA Best Practices
The American Institute of Architects (AIA)	AIA Contract Documents
Process Industry Practices	Practices

The Construction Industry Institute (CII) is one of the most well-known organizations that promote best practices within the construction industry domain. CII defines a best practice as “a process or method that, when executed effectively, leads to enhanced project performance”. CII criteria to define a practice as a Best Practice are as follows (*Benchmarking & Metrics Implementation Toolkit*, 2004): 1) there is a defined process and method with steps and activities, 2) comprehensive research has proven the value of the practice, and 3) the industry has accepted and is using the practice. Table 2-2 depicts a summary of CII best practices.

Table 2-2: CII Best Practices

1. Advanced Work Packaging	10. Materials Management
2. Alignment	11. Partnering
3. Benchmarking & Metrics	12. Planning for Modularization
4. Change Management	13. Planning for Startup
5. Constructability	14. Project Risk Assessment
6. Disputes Prevention & Resolution	15. Quality Management
7. Front-end Planning	16. Team Building
8. Implementation of CII Research	17. Zero Accidents Techniques
9. Lessons Learned	

Constructions Owners Association of Alberta (COAA) is another renowned organization in developing and promoting construction industry best practices. Table 2-3 presents the list of COAA best practices. Construction performance best practice includes subcategories of benchmarking, workforce planning, advanced work packaging, rework reduction, project productivity, and modularization.

Table 2-3: COAA Best Practices

1. Safety	4. Construction Performance
2. Workforce Development	<ul style="list-style-type: none"> • Benchmarking • WorkFace Planning • Advanced Work Packaging • Rework Reduction • Project Productivity • Modularization
3. Contracts	

Companies implementing best practices consistently report higher profits, increased customer satisfaction, and improved safety and productivity. CII criteria for defining a practice as a best practice implies that most known best practices in the construction industry are process based or can be defined as processes. Defining the essence of a best practice as a process has several

advantages for automation and integration of best practices into information systems. A promising solution for facilitating more effective and consistent conformance with the best practices lies in the employment of processes, process models, and workflow engines.

2.2.1 Best Practices as a Form of Knowledge

Although it is difficult to define knowledge, there are some widely accepted classifications for it. Knowledge hierarchy or DIKW (Data, Information, Knowledge, Wisdom)s pyramid defines the relationship between data, information, knowledge and wisdom. In this classification, information is defined in terms of data, knowledge is defined in terms of information, and wisdom is defined in terms of knowledge. Table 2-4 shows a summary of knowledge hierarchy classification, its definitions and outcomes (Anand & Singh, 2011).

Table 2-4: Summary of Knowledge Hierarchy

LEVEL	DEFINITION	OUTCOME
Wisdom	Applied knowledge	Judgment
Knowledge	Organized information	Understanding
Information	Meaningful and useful data	Comprehension
Data	Raw facts and figures	Memorization

Another classification of knowledge relies on the difference between explicit and tacit knowledge. Explicit or codified knowledge is the knowledge that is easy to identify, store, and retrieve (Wellman 2009), such as that found in documents, texts, and databases. This is the type of knowledge most easily handled by knowledge management systems. Tacit or non-codified knowledge, on the other hand, is the knowledge that is largely intuitive, experienced-based and hard to codify, such as the knowledge to skillfully ride a bike, or play a piano. Although shades of these skills can be described in texts or documents, no one can learn them merely by reading those documents. Tacit knowledge is associated with the knowledge embedded in people based on their cultural beliefs, values, attitudes, mental models, etc. as well as their skills, capabilities and expertise (Botha, Kourie, & Snyman, 2008).

In practices, knowledge is a mixture of tacit and explicit elements (Botha et al., 2008). Data are more associated with explicit knowledge and as we go up through the knowledge hierarchy, there exist a stronger association with tacit knowledge (Figure 2-1).

Knowledge management is the process of capturing, sharing, and effectively using organizational knowledge (Botha et al., 2008) and knowledge management systems are generally IT-based systems that facilitate the best use of knowledge. Knowledge management systems might have different approaches and methods to fulfil this objective; however, almost all of the recent versions use semantic technology to more precisely categorize and describe the meaning, and define the relationship among any piece of information.

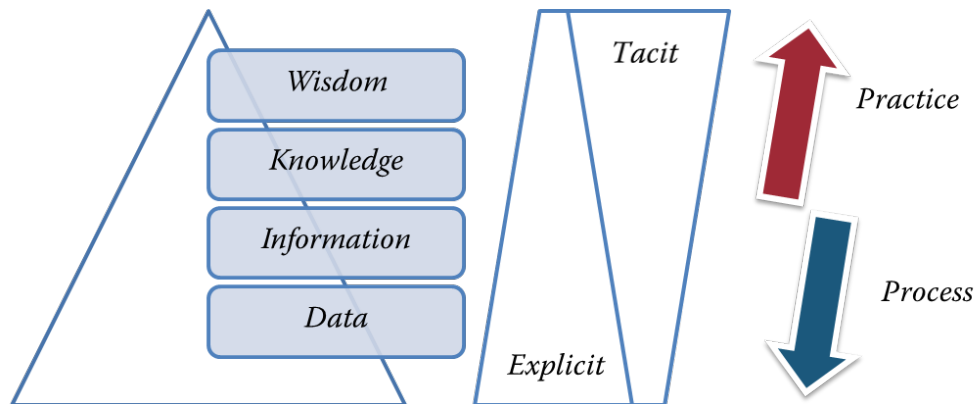


Figure 2-1: Knowledge Hierarchy and Its Association with Tacit & Explicit Knowledge, and with Practice and Process

2.3 Process Management

The required tools used in the context of information systems to capture, model, and analyze different types of information have also evolved from data modeling to process modeling tools and techniques. The main purpose of data models is to support the development of information systems by providing the definition and format of data. Process models, on the other hand, are functional models describing process activities, and their associated properties, sequences, and execution constraints. Data and process models are used for proper communication between business and technical people in the context of business process management.

Business process management and workflow engines have been used to provide automation, integration, and interoperability for information systems in many sectors, particularly banking, healthcare, and manufacturing. Automation – the utilization of electronic or computerized tools to make a task more efficient – is inherent in utilization of IT tools. Integration – the ability of sharing information from multiple sources between two or more systems – typically exists within software

packages produced by a specific vendor. However, a vendor specific integrated system is generally not able to share information with integrated systems from other vendors (Shen, 2010).

While integration enables two or more systems to seamlessly work together, interoperability meets the same objective by following a standard protocol for the interaction of these systems. Therefore, interoperability provides the additional advantage of allowing other systems to interact with these systems by adopting a protocol.

Business process management is the holistic approach for managing business processes within an organization ranging from the design, modeling, and execution stages to monitoring and optimization. Business process management is based upon explicit representation of business processes, their activities, and their execution constraints (Weske, 2012). A business process management system is a software system for coordination and enactment of the activities involved in business processes.

2.3.1 Process – Definition and Levels

A process is a series of well-defined inter-related steps, which delivers repeatable, predictable results. Key features of a process include 1) predictable and definable inputs, 2) linear, logical sequence, 3) clearly definable set of activities, and 4) predictable, desired outcome (L. L. Lee, 2005). A business process consists of a set of linked activities that are performed in coordination to serve a business goal such as delivering a product or service to a customer. An activity is typically considered as a major unit of work comprising more detailed steps called sub-activities. Whenever an activity is considered as the smallest unit of work, it is usually called a task.

A business process can be performed manually or can be automated through an information system. For an automated process, the inputs, outputs, and steps involved should be clearly defined; and to implement a process in a workflow management system it should be defined in a standard process modeling language. Typically, automated processes include both automated and manual activities. Request for Information (RFI) and Contract Management (CM) processes are examples of such processes. Processes in which all of their activities are automated are called fully automated processes such as buying processes in Amazon or eBay.

Business processes can be classified, from high level to more structured, into four levels: 1) goals and strategies, 2) organizational business processes, 3) operational business processes, and 4) implemented business processes, as illustrated in Figure 2-2. Informal modeling tools such as plain text and generic diagrams are used for representing higher level processes such as goals and strategies and organizational business processes. Formal modeling techniques and standards are used for modeling processes of the operational level. Implemented processes are the executed instances of operational processes which include execution and more technical details.

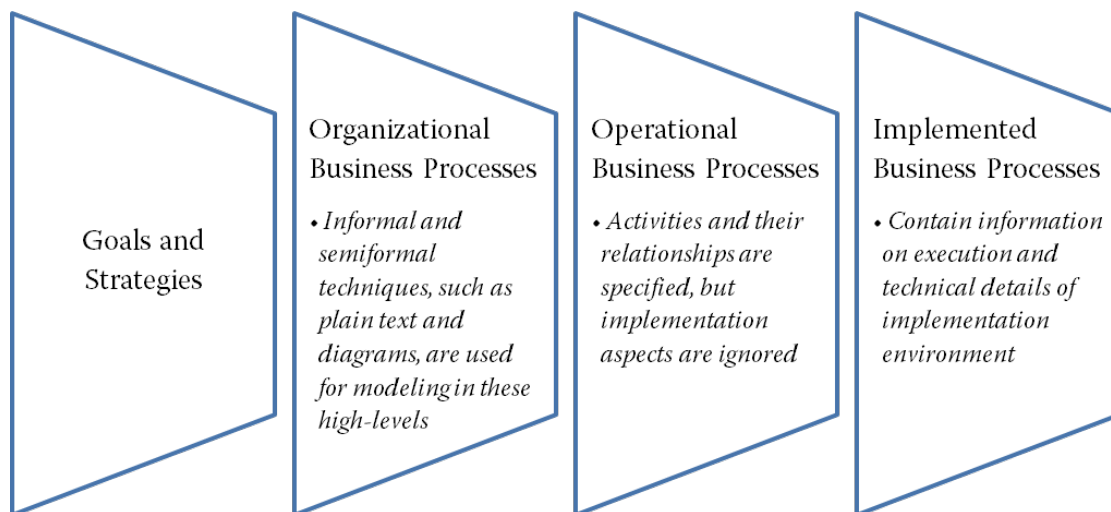


Figure 2-2: Classification of Business Processes (Weske, 2012)

A process in which the sequence of activities and their execution constraints are completely defined is called as structured process. The lower level business processes – operational and implemented levels – are typically defined as structured processes.

In any organization, business processes are part of the knowledge management system and are intangible assets. Unlike tangible assets (resources) such as materials, machinery, and infrastructure, intangible assets evolve over time and cannot easily be acquired. Business process management facilitates identifying, analyzing, and improving business processes within organizations.

2.3.2 Process Modeling

Process modeling is the representation of a process in an appropriate format in order to design, analyze, and improve it. It is the main technical stage in the process design phase. Process

modeling techniques are used as the means of communicating the structure and details of a process among process stakeholders.

Typically, business processes have three distinct groups of stakeholders each with different viewpoints: 1) managers and business administration people, 2) business analysts, and 3) software developers (Figure 2-3). Business administration people typically use informal and semiformal techniques such as diagrams and plain text to discuss about business processes. They deal with organizational level business processes which are high-level business processes.

Transforming high-level description of business processes into a more structured and formal definition is the responsibility of business analysts. They use formal business process modeling tools such as standard business process models to represent processes in a structured format. Software developers then use modeling and programming languages to implement business processes in a software platform and link it to enterprise information systems.

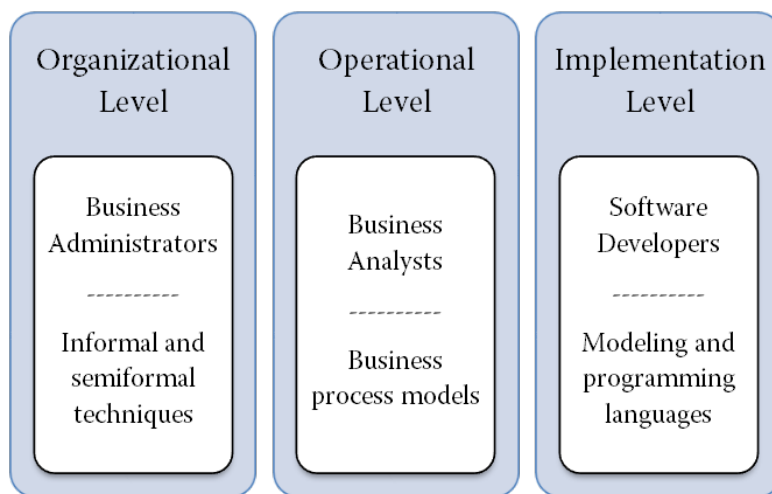


Figure 2-3: Typical Users and Tools for Each Process Level

Since the background and interests of the stakeholders are different, they look at the same process from different viewpoints and use different conceptual levels. Communication problems between them, are thus, expected and normal. More recent process modeling tools, such as Business Process Modeling and Notation (BPMN), are trying to bridge this communication gap.

Process modeling tools are used to transform informal descriptions of high-level organizational processes into formal operational level process definitions using standard modeling notations. The resulting model is called a process model or a functional model.

2.3.3 Process Modeling Tools and Standards

Several process modeling tools have been developed for modeling business processes. Process modeling can either be performed by representing a process using structured graphical notations or by representing the semantics of the process using modeling languages. A process model is typically defined as the graphical structured representation of a process, because using graphical notations is more convenient for communicating, reengineering, and improving of processes. Recent modeling tools such as XPDL and BPMN support both a graphical notation and a modeling language. A classification of most popular modeling tools is presented in Table 2-5.

Table 2-5: Process Modeling Tools

CLASSICAL	MORE FORMAL	MOST RECENT
Flowchart 1920s	Petri nets 1960s	UML 1997
Functional Flow Block Diagram (FFBD) 1950s	Workflow patterns YAWL	XPDL 2002 BPEL 2004
Data Flow Diagram (DFD) 1970s ICAM DEFinition (IDEF0) 1970s	Graph-Based Workflow Language	BPMN 2004 (BPMN 2.0 2011)

Flowcharts are among the oldest process modeling graphical notations. They offer a simple notation for process modeling which is the basis for developing many subsequent modeling notations. Flowcharting techniques are still the preferred method of high level process modeling for managers and business administration people.

Petri net is a mathematical modeling language with clear and well defined semantics. Petri net offers a graphical notation and a precise mathematical definition. It has been used in several academic publications for discussing process behaviors. While petri net is very useful in expressing simple types of processes, more complex processes such as business processes, require more advanced structures (Weske, 2012). Several other modeling languages such as Workflow Patterns, Yet Another Workflow Language (YAWL), and Graph-based Workflow Languages are enhancements over the traditional petri net, adding more concepts and features for modeling more complex processes.

The rise of new software development paradigms and the need for standardization of modeling tools for modeling more complex processes led to development of modern modeling tools, such as Unified Modeling Language (UML), XML Process Development Language (XPDL), Business

Process Execution Language (BPEL), and the currently emerging Business Process Modeling and Notation (BPMN).

Unified Modeling Language (UML) is a general-purpose modeling language by Object Management Group for object oriented software development. UML offers 14 types of diagramming notations for different modeling purposes in which Activity Diagrams are specifically used for process modeling. Business Process Execution Language (BPEL) is a standard executable language by OASIS. Its focus is exclusively on the executable aspects of business processes and does not offer any graphical notation. XML Process Definition Language (XPDL) is another standard format for interchanging business process definitions between different products using XML syntax, developed by the Workflow Management Coalition (WfMC). It is designed to exchange both the graphics and the semantics of a process definition. In 2004 the WfMC endorsed BPMN, and since then XPDL has been extended specifically with the goal to represent all of the concepts present in a BPMN diagrams in XML format.

Business Process Modeling and Notation (BPMN) is the most promising process modeling standard. It has been designed by Object Management Group (OMG) with the aim of identifying best practices of existing modeling tools and combining them into a widely accepted, easy to use language. BPMN aims at supporting all the process abstraction levels, from business organizational level to implementation level, and thus, bridging the gap between process modeling and implementation (Weske, 2012). The same process model in BPMN may encompass different levels of details, each useful for a particular group of stakeholders, from business administration people to business analysts and software developers. BPMN defines three levels of process modeling conformance. Descriptive level, useful in high-level modeling, only includes visible elements and attributes; analytic level includes descriptive and a minimal subset of supporting process attributes; and common executable offers the elements required for execution of process models. The current version is BPMN 2.0.2 introduced on January 2014.

2.3.4 Process Specialization

Object-oriented analysis and design methodologies that are originated from object-oriented programming are now being used for design and implementation of systems such as information systems. In the object-oriented design approach, a class represents a set of objects with a common

structure and behavior. An object is an instance of a class with a set of attributes and methods. The state of an object is determined by the value of attributes. Methods are operations on an object that can change the state of an object by changing the value of attributes.

Abstraction and inheritance are two important concepts in object-oriented design that facilitate modularity and reuse of system components. Abstraction is the process of representing the right amount of detail and hiding unnecessary implementation or background details. The inheritance mechanism allows a subclass to inherit features of a superclass so that the subclass has the same features as the superclass, but it typically includes some additional features (Basten & van der Aalst, 2001).

Object-oriented concepts have been used to provide abstraction and inheritance functionalities for process models. An example of such efforts is the MIT Process Handbook which is a repository of more than 5000 organizational processes associated with various business models. This repository is a classification of processes by two dimensions: parts and types. Any process can be specialized either from its uses to its parts or from its general type to a more specialized activity. The scope of processes in the MIT Process Handbook is general business processes and they have been organized in such a way to be easily used in the design of new processes or for reengineering of existing business processes.

2.3.5 Process vs. Practice

Distinguishing the difference between process and practice is important. A process, as discussed, is a series of well-defined inter-related steps, which delivers repeatable, predictable results. A process, thus, is typically used in routine circumstances in which repeatable, predictable results are required. Each necessary step is codified in detail and there is no spontaneous decision making involved. A practice, on the other hand, is a frequently repeated act, habit or custom that needs a recognized level of skill to be performed. It is an un-codified knowledge that results from human experience and improvisation (L. L. Lee, 2005).

While a practice is still a series of steps, the steps are roughly defined, and the details of how to perform each step is left to the experts who perform them based on their knowledge, experience, skill, and judgment. Practices, thus, are more suitable for dealing with uncertain situations with

uncommon unique results (“IT Catalysts,” 2013). Table 2-6 summarizes key differentiators of processes and practices.

Table 2-6: Process vs. Practice

PROCESS	PRACTICE
Series of well-defined steps	Series of steps, but loosely defined
Deliver repeatable, predictable results	The specifics are left to the practitioners
Well-suited to mass production	Well-suited to the creation unique results, dealing with ambiguous situations, and especially in competitive arenas
Includes clear steps and details for tasks	Not necessarily have a clear sequence and details for tasks

A best practice is a form of knowledge with the consensus on providing higher benefits, when used properly. Well-known best practices are typically promoted by renowned organizations in a certain field, and are grounded on the result of collective wisdom, experience, research, careful investigations, and extensive industry use and validation.

2.3.6 Workflow vs. Process

Workflow and process are similar terms and, in certain situations, might be used interchangeably. However, workflow implies a more specific concept than process. While any well-defined interconnected steps with an expected result can be called a process, in a workflow the focus is on the piece of work or information that is being passed through initiation to completion. Therefore, a workflow associated with a particular process might not be involved with all the details that are important for completion of the process, such as recording to a database or calling a web-service, but is more dedicated to the flow of work through all steps. A workflow thus can be defined as an outline or blueprint of a process.

Although a workflow is typically an organizational level process, it can include operational and implementation details whenever required, and thus a workflow specification can be defined with different abstraction levels. The abstraction level depends on the intended use of the workflow specification. For instance, a workflow specification for a process might be defined in a higher conceptual level required for understanding, evaluating, and redesigning the process. The same process might be captured in another workflow specification with a lower level of abstraction, and include the execution details necessary for workflow implementation (Georgakopoulos, Hornick, & Sheth, 1995). A summary of process and workflow differences are presented in Table 2-7.

Table 2-7: Process vs. Workflow

PROCESS	WORKFLOW
A process is a series of well-defined inter-related steps	A workflow can be considered an outline of a process
A process is modeled using modeling tools and implemented by coding the steps	The flow of work in a workflow can be updated without changing underlying code
A process can be modeled with different abstraction levels: organizational, operational, and implementation levels	The focus is on organizational details, but can include operational and implementation-level details
The focus is on steps of work	The focus is on the flow of work
A programmer typically implements a process	An analyst typically can modify the steps and update the flow of a workflow

2.4 Information Management Systems

Conventional data-aware information systems evolved around centralized database management systems. Today's process-aware information systems facilitate interaction and collaboration of stakeholders via distributed systems. Examples include: electronic document management systems (EDMS), workflow management systems (WfMS), content management systems (CMS), enterprise resource planning (ERP), electronic product and process management systems (EPPMS), and Business process management systems (EDMS). WfMS and EPPMS are discussed in this study more than others.

2.4.1 Workflow Management Systems (WfMS)

Workflow management and workflow specification are concepts tightly related to business process management and process modelling; their approach is rather different. Workflow management involves the automation of processes which are comprised of human and machine-based activities (Hollingsworth, 1995) and focuses on the flow of information or work among participants. A workflow specification is an abstraction of a process that might not be concerned with all the details of a task, but in any case it is concerned with the inter-relationship, the inputs and outputs, and the externally visible behavior of tasks (Krishnakumar & Sheth, 1995).

Automation of business processes partly relies on the coding of software developers for embedding business processes into information systems. Originally, any modification to the process logic, the sequence of activities, and the execution constraints of a process was affecting the programming code and required software developer's attention. The introduction of object-oriented

programming concepts facilitated the separation of process logic modifications from the programming code, and led to the emergence of workflow driven systems.

In a workflow management system, features of an application, or tasks of a process, are defined as steps in a workflow, and therefore, the behavior of the system can be modified through changing the steps without any modification to the programming code. Workflow technology, thus, provides separation of business process logic from IT operational support (Hollingsworth, 1995).

A workflow engine is responsible for managing and enacting tasks within workflow specifications according to their execution constraints and organizational predefined rules. The execution constraints of a process are typically defined as properties or attributes of tasks in the workflow specification. The Workflow Reference Model (Hollingsworth, 1995), developed by the Workflow Management Coalition (WfMC), defines the general specifications of a workflow management system, and still is a key reference for developing workflow management systems and their interfaces. Workflow management systems facilitate more convenient design and implementation of processes with less involvement in programming details.

2.4.2 EPPM System

An Electronic Product and Process Management (EPPM) system is a workflow management system specifically designed for managing large-scale construction projects. A workflow engine at the heart of an EPPM system facilitates enactment of workflow processes; a document management system supports several types of files and enables sharing and modifying various types of documents; and a collaboration management system enables project delivery by collaboration among several stakeholders. In addition, the kernel of an EPPM system typically offers services, such as format management, version control, indexing, search, security, and publishing. EPPM systems store and manage various types of information regarding the lifecycle of a project from inception and planning to execution and startup. These systems not only facilitate enactment of processes via workflow engines, but they also facilitate interaction of process stakeholders and tracking and auditing of process steps. For example, change management, deliverables management, or interface management processes typically involve the interaction of several stakeholders, such as contractors, sub-contractors, suppliers, consulting firms, and the

owner(s). An EPPM system provides the infrastructure for defining, modifying, enacting, and auditing such processes.

2.5 Interoperability – Definition and Levels

The idea of interoperability started from a pure software problem in the middle of 90's, and it is taking on a broader meaning and wider application to cover the many knowledge areas, dimensions and layers of single and collaborating enterprise (Chen, Vallespir, Daclin, & others, 2008). In the context of this research, interoperability can be defined as the ability to effectively, accurately, and consistently communicate and exchange information, within different information technology systems (Gibbons et al., 2007). It provides a way for two or more systems to seamlessly work together by automatic and timely exchange of information, and prevents the manual steps otherwise needed to transform information from one system to the other. However, more generally, the interoperability is still an imprecise concept with many definitions and connotations to different people in different sectors and domains (Chen, Vallespir, et al., 2008).

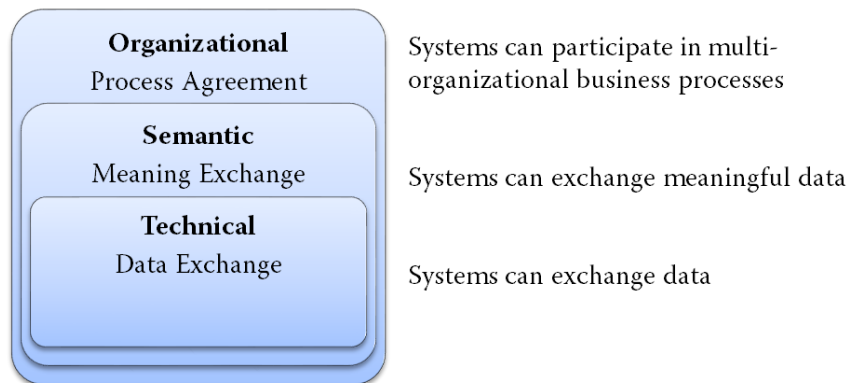


Figure 2-4: Three Levels of Interoperability (Lewis, 2013)

A review of the literature and a survey across all industries by the Electronic Health Record (EHR) Interoperability Work Group (Gibbons et al., 2007) identified 65 definitions for interoperability from standards development organizations, health care organizations, professional societies, and government agencies. In spite of substantial differences in the definitions, three principal levels of interoperability were identified: technical, semantic, and process interoperability (Figure 2-4). Technical interoperability enables data exchange among systems; semantic interoperability enables exchange of meaningful data; and organizational or process interoperability enables coordination of work processes through participation in multi-organizational business processes

(Lewis, 2013). Process interoperability is also called workflow or social interoperability (Gibbons et al., 2007).

Other variations of classification for interoperability levels have since emerged. For example, the European Telecommunications Standards Institute (ETSI) has introduced a syntactic level between the technical and semantic levels (Veer & Wiles, 2008). Based on the ETSI classification, Kubicek and Cimander (Kubicek & Cimander, 2009) have summarized what each layer of interoperability aims at, what is exchanged, by which standards, and the state of maturity of each layer (Table 2-8).

Technical interoperability is associated with communication protocols and the infrastructure – hardware or software – needed for those protocols to operate. Syntactic interoperability is typically associated with data formats (Veer & Wiles, 2008). Technical and syntactic interoperability rely on established standards such as TCP/IP for data transfer and XML for data exchange. Technical and syntactic interoperability facilitates the exchange of clearly defined classes of data, whereas semantic interoperability enables recognition and interpretation of the data exchanged. The concepts and methods for semantic interoperability are available, but are not standardized yet. For organizational interoperability, however, there is no consensus on a framework of what should be standardized (Kubicek & Cimander, 2009).

LAYER OF INTEROPERABILITY	AIM	OBJECTS	SOLUTIONS	STATE OF KNOWLEDGE
TECHNICAL	Technically secure data transfer	Signals	Protocols of data transfer	Fully developed
SYNTACTIC	Processing of received data	Data	Standardized data exchange formats, e.g. XML	Fully developed
SEMANTIC	Processing and interpretation of received data	Information	Common directories, data keys, ontologies	Theoretically developed, but practical implementation problems
ORGANIZATIONAL	Automatic linkage of processes among different systems	Processes (Workflows)	Architectural models, standardized process elements (e.g. SOA with WSDL, BPML)	Conceptual clarity still lacking, vague concepts with large scope of interpretation

Table 2-8: Four Levels of Interoperability

While the main focus of technical, syntactic, and semantic interoperability is data – e.g., data transfer, exchange, and meaning – the focus of organizational interoperability is processes – e.g., process and workflow alignment – and how the work is being performed. Process interoperability is a higher level of interoperability, and it should be regarded as indispensable once other layers of interoperability have been achieved.

Process interoperability is associated with process/workflow management, and deals with the successful integration of advice/alerts into data presentation and workflows, and/or the deployment of workflow resources in conformance with a plan or protocol. Process interoperability is critical in successful implementation of and the use of IT systems that extend over multiple organizations. For instance, in healthcare, lack of process interoperability is cited as a likely reason that more than fifty percent of health information technology implementations fail to meet expectations (Gibbons et al., 2007).

2.6 Interoperability in AEC/FM Domain

Interoperability among construction industry IT systems, such as computer-aided design and engineering (CAD/CAE) and building information modeling (BIM), has been one of the major themes of research and development in the domain of architecture, engineering, construction, and facilities management (AEC/FM) (Froese, 2003), and has had a commensurate impact on facilitating collaboration and improving productivity of construction projects. However, in these systems the focus has mainly been on data-oriented issues, with little or no primary emphasis on process models.

Data models are used in these systems to facilitate representation of two primary types of structured information: (1) geometric data and information relating to geometry of objects, and (2) product data, associated properties and related information. Although these systems to some extent can exchange non-geometric (e.g. information about the design process, construction process, cost estimating and material take off, etc.), representation of the design artifact itself is still generally limited to geometry. (Szykman, Fenves, Keirouz, & Shooter, 2001).

Interoperability can be viewed from two different perspectives: data interoperability and process interoperability. Data interoperability is concerned with the accurate interpretation and

understanding of the information exchanged. In the construction industry, data exchange techniques and data interoperability standards have been on the focus and improved substantially over the years through employment of data modeling techniques. The most well-known data modelling and interoperability standards in the domain on AEC/FM are ISO 16739 and IDP 15926 data models. ISO 16739 or Industry Foundation Classes is the data model underlying the BIM technology and ISO 15926 is the standard data model for integration, interoperability, and life-cycle data exchange in process plants, including oil and gas production facilities.

Process interoperability, however, ensures seamless communication between different systems by developing a shared understanding of their process constructs (Khan et al., 2013). Process conformance and interoperability, within the construction industry, is an emerging need especially with increased use of the new generation of workflow-driven software platforms such as EPPM systems.

2.7 The Knowledge Gap

The process-oriented approach of managing projects and businesses is a well-established approach which includes several innovative tools and techniques. Some research studies investigated the differences between processes and practices and the reasons a practice cannot completely and effectively be transformed into a process. Others described the core structure of a process and offered a framework for process customization. However, the literature lacks a systematic approach for transformation of a best practice into a process that methodically defines how and which components of a best practice can be transformed into a process implementable into workflow management systems.

In addition, there is a knowledge gap regarding improving process conformance and interoperability by defining a unique core structure for common processes in the construction industry, based on the best practices in this domain. Developing a methodical approach to integrate best practices into workflow management systems in such a way as to provide conformance and interoperability is the main objective of this research, in order to address these knowledge gaps, and improve capital project performance.

Chapter 3

Industry Foundation Processes (IFP)

3.1 IFP Modeling System and IFP Processes

Industry Foundation Processes (IFP) is a process modeling system that facilitates integration of core processes of known best practices in the construction industry into workflow management systems, and promises to improve projects' process conformance and interoperability. IFP processes are defined in this study as workflow templates with essential activities and minimal features that can be customized for specific types of projects, and implemented based on projects conditions and requirements. Through IFP model implementation, the EPPM system not only supports best practices conformance and process interoperability, but it also provides automation and integration of other systems processes and services, thus facilitating improved project performance.

IFPs¹ are defined as structured processes so that the sequence of activities and their execution constraints are fully defined. They focus on the flow of information or work while abstracting from execution constraints, such as data dependencies and resource constraints. They are defined in their simplest form, containing all the essential steps, but with no extra or redundant activity. As such, they are general enough to be extendable to many situations, yet simple and streamlined. The idea of the IFP system is inspired by the concepts of abstraction, inheritance, and modularity in object-oriented programming languages (OOP), and its name has a connotation with the Industry Foundation Classes (IFC) data model.

IFPs are abstracted to operational-level details, with the focus of enactment through workflow management systems. The workflow inheritance concept enables IFP workflow processes to be customized to more specific and more complex processes in a controlled manner to conform to particular types and characteristics of projects, while not losing their core structure. The IFP modeling system may be defined for many common construction industry processes, such as

¹ Throughout this thesis, the "IFP" acronym for Industry Foundation Processes refers to the IFP modeling system as well as to a single IFP process. The plural form "IFPs" refers to more than one IFP process.

change management, contract management, materials management, and deliverables management, as simple structured processes that incorporate the essence of best practices.

Application of the IFP system offers several practical advantages. It promotes adoption of best practices, provides a standard core structure for implementation of common processes, facilitates more consistent implementation of workflow processes in different projects, and brings visibility to the core structure of complex processes. It also improves process conformance and interoperability. This system facilitates integration of best practices into workflow management systems, and supports their consistent implementation throughout project lifecycle and from project to project. It can be used to efficiently implement and manage systems of customized interoperable processes that conform to the best practices, and thus support improved project performance.

3.2 Approaches of Developing IFP Processes

The IFP system facilitates integration of core processes of known best practices in the construction industry into workflow management systems, in order to provide a more consistent and scalable method for adoption of construction industry best practices, throughout the lifecycle of each project, and from project to project. Best practices are a form of knowledge that are based on the lessons learned and the experience gained from previous projects. They facilitate reuse of experience within the construction industry domain by suggesting an improved way of organizing and performing construction management activities.

Best practices typically include one or more processes or can be defined as one or more high-level processes that represent the main steps of performing the related work. However, such high-level processes cannot be directly implemented as workflow processes into workflow management systems. The steps offered in best practices does not necessarily include a well-defined sequence, the execution constraints for performing the associated work are not explicitly defined, and the role and the responsibilities of the actors might not be clearly defined.

Accordingly, two principal approaches are proposed for deriving foundation processes (Figure 3-1): (1) a bottom-up approach in which the common core structure of different implementations of a construction workflow process is identified and extracted, and is used as the

basis for developing an IFP, and (2) a top-down approach in which foundation processes are defined as structured processes in accordance with the existing best practices in the construction industry. Although these two approaches are different in methodology and can be used separately, using a combination of both approaches, if applicable, is recommended for the best outcome.

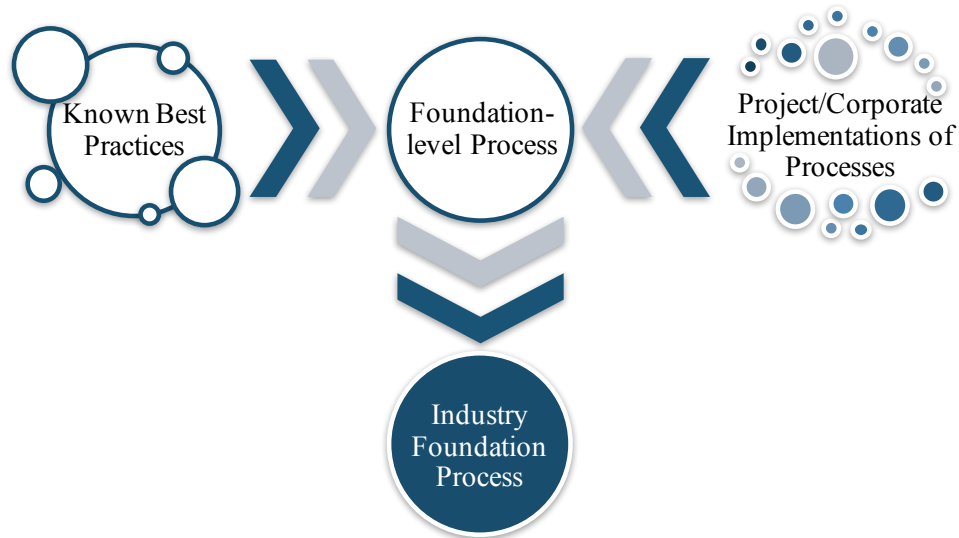


Figure 3-1: Approaches of Developing IFP Processes

The former approach is useful for workflow processes that have been used in different projects, and their implemented versions are available. This approach requires employing business process analysis tools and process modeling techniques to compare different implementations of a process and extract the common core of those processes as a basis for deriving a foundation process. The latter is used to define a new or distinct workflow process for performing a specific operation in accordance with the established best practices. It involves exploring well-known construction industry best practices, developing high-level organizational processes that include the main steps for adopting those practices, transforming the organizational processes based on the roles and responsibilities of actors into structural processes implementable into workflow management systems, and defining IFPs based on the core structure of the structured processes. These approaches are discussed in more detail in sections 3.3 and 3.4, respectively.

3.3 Extracting the Common Core of Implemented Processes

Sending or receiving engineering documents as transmittals or submittals, reviewing and approving design documents, requesting further information, managing change orders, and

managing contractual obligations are examples of common activities that have been intrinsic components of construction projects for years. In more recent years, these common activities have been automated via workflow management systems and have been implemented as structured workflow processes such as inbound and outbound transmittals (IT & OT), design review (DR), request for information (RFI), change request (CR), and contract management (CM).

Common workflow processes are being implemented differently in each organization due to several different factors, such as organizational culture, structure, governance, established communication channels, and available resources which are largely categorized as enterprise environmental factors. Workflow processes are implemented differently in each project depending on the type, requirements, resources, geographical distribution, and other conditions of that particular project. Such processes are part of organizational process assets. Organizational process assets are defined by Project Management Institute (PMI) as “plans, processes, policies, procedures, and knowledge bases specific to and used by the performing organization” and are grouped into (1) corporate knowledge base, and (2) processes and procedures (Project Management Institute, 2013).

Workflow processes that are used in capital construction projects are part of intangible assets of the performing organization. Such processes are carefully crafted by experts for a specific purpose and have typically been subject to several cycles of process improvement since their creation. Each update refines the process in a certain way and creates an improved version with a particular version number. Ultimately, the process possesses the most suitable activities, flow, and details for performing that specific work, and represents a best method of doing that work in that organization or project.

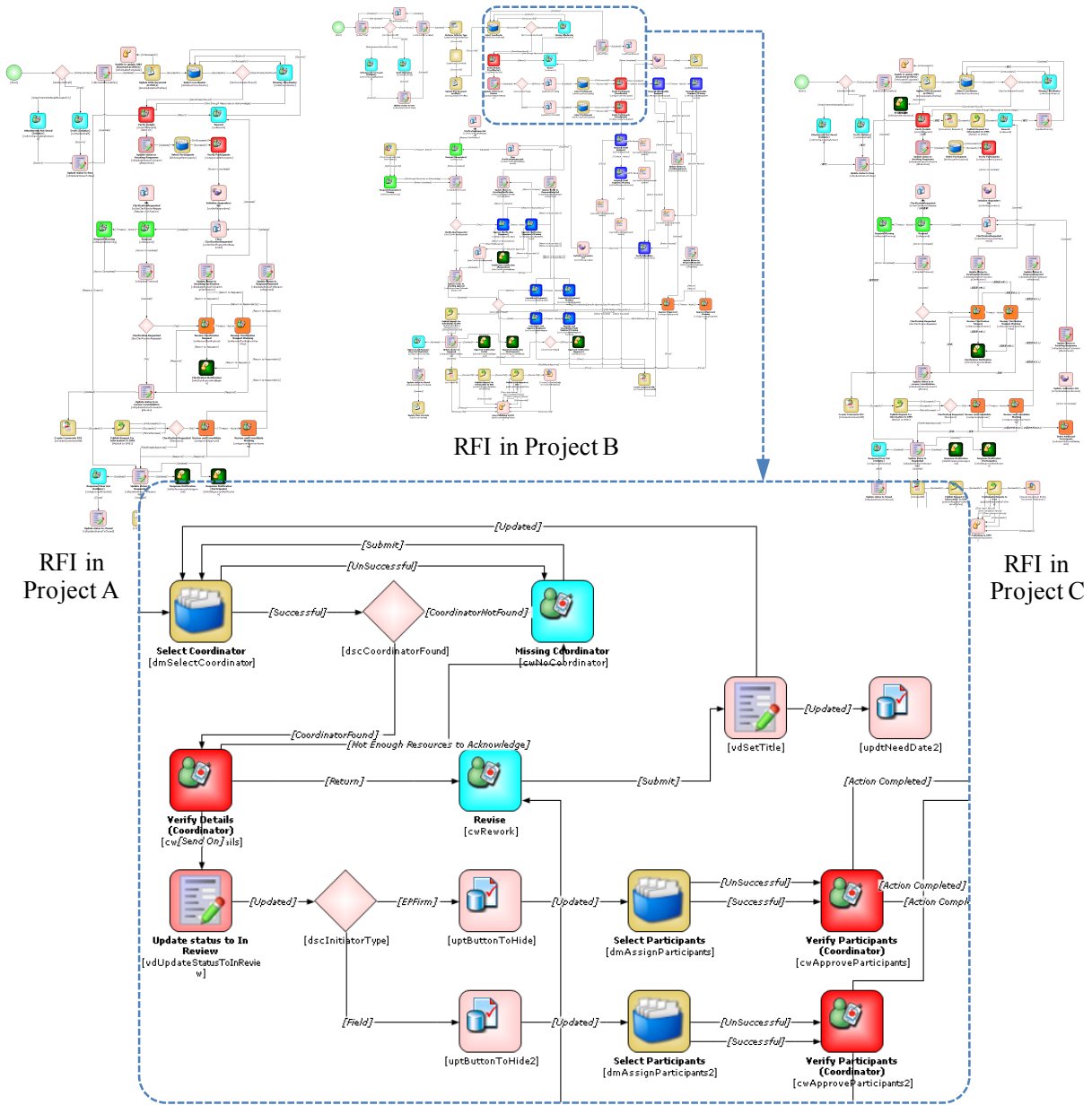


Figure 3-2: Different Implemented Versions of the RFI Process in Skelta Software Format

For instance, Figure 3-2 demonstrates three implementations of the Request for Information (RFI) workflow process in three different large-scale construction projects. The magnified portion demonstrates some of core activities in one of them. RFI workflow is a method of requesting a design clarification, field construction clarification, or to provide supplemental instructions from either the project management team, or any company engaged in a construction project. Each of the implementations in Figure 3-2 include different versions. For example, the figure in the center, which is partly magnified and has been used in a recent mega-construction project in Canada,

encompasses eight versions. Each version is slightly different and is the result of a process improvement effort during the lifecycle of the project.

Although the implementation of a common process, such as the RFI, varies from organization to organization, and is unique in each project, their common core structure is not very different, when those implementations are compared in a higher abstraction level in which the implementation level differences, such as technology and platform-specific relations, and execution details are ignored. Process analysis tools and process modeling techniques can be used to analyze the implemented versions of a process, and to compare their structure in a higher level of abstraction, and to extract their common core structure.

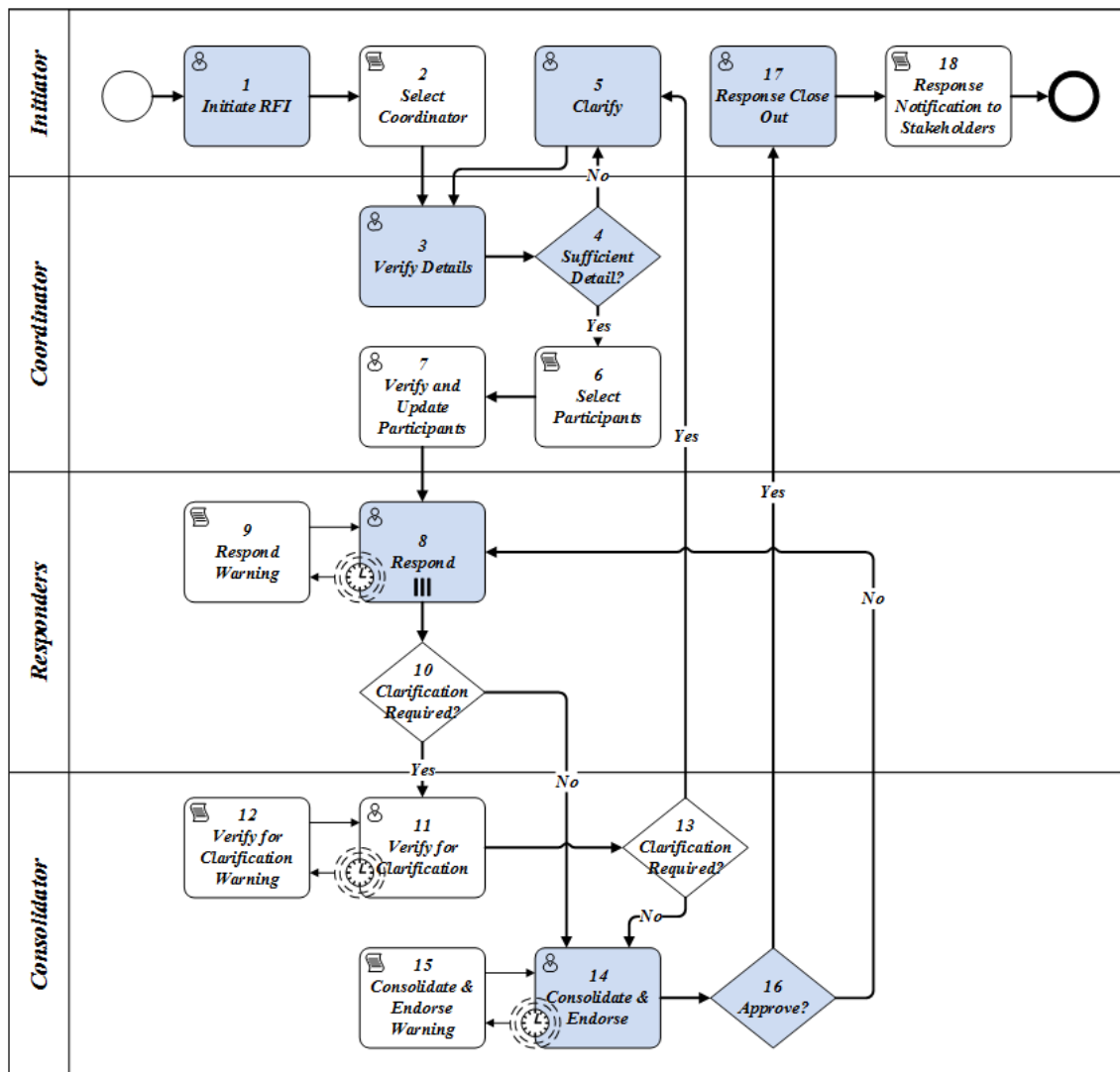


Figure 3-3 (a): High-Level Representation of the RFI Workflow Process in Project A

Figure 3-3 (a), (b), and (c) demonstrate a higher level process model representation of the same RFI processes shown in Figure 3-2. In these process models several execution and implementation level details, such as initializing variables, updating the status in each step, handling missing coordinator, publishing the request to the document management system (DMS), initializing the response list, publishing the attachments to the DMS, creating pdf files, etc. have been abstracted from the model and only the key relevant activities have been shown. This higher level representation enables more explicable analysis and comparison of these workflow processes.

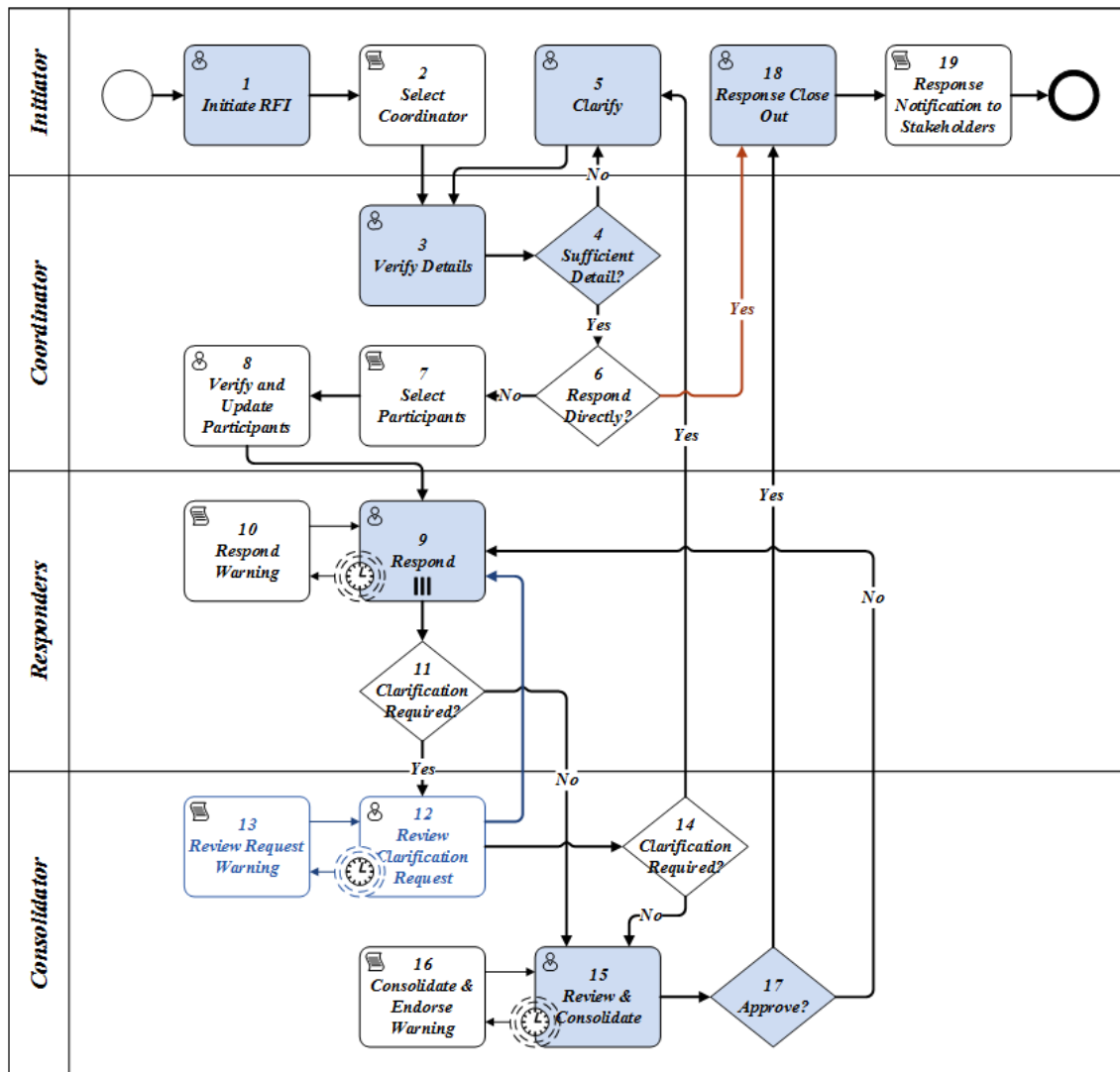


Figure 3-3 (b): RFI High-Level Representation of the RFI Workflow Process in Project B

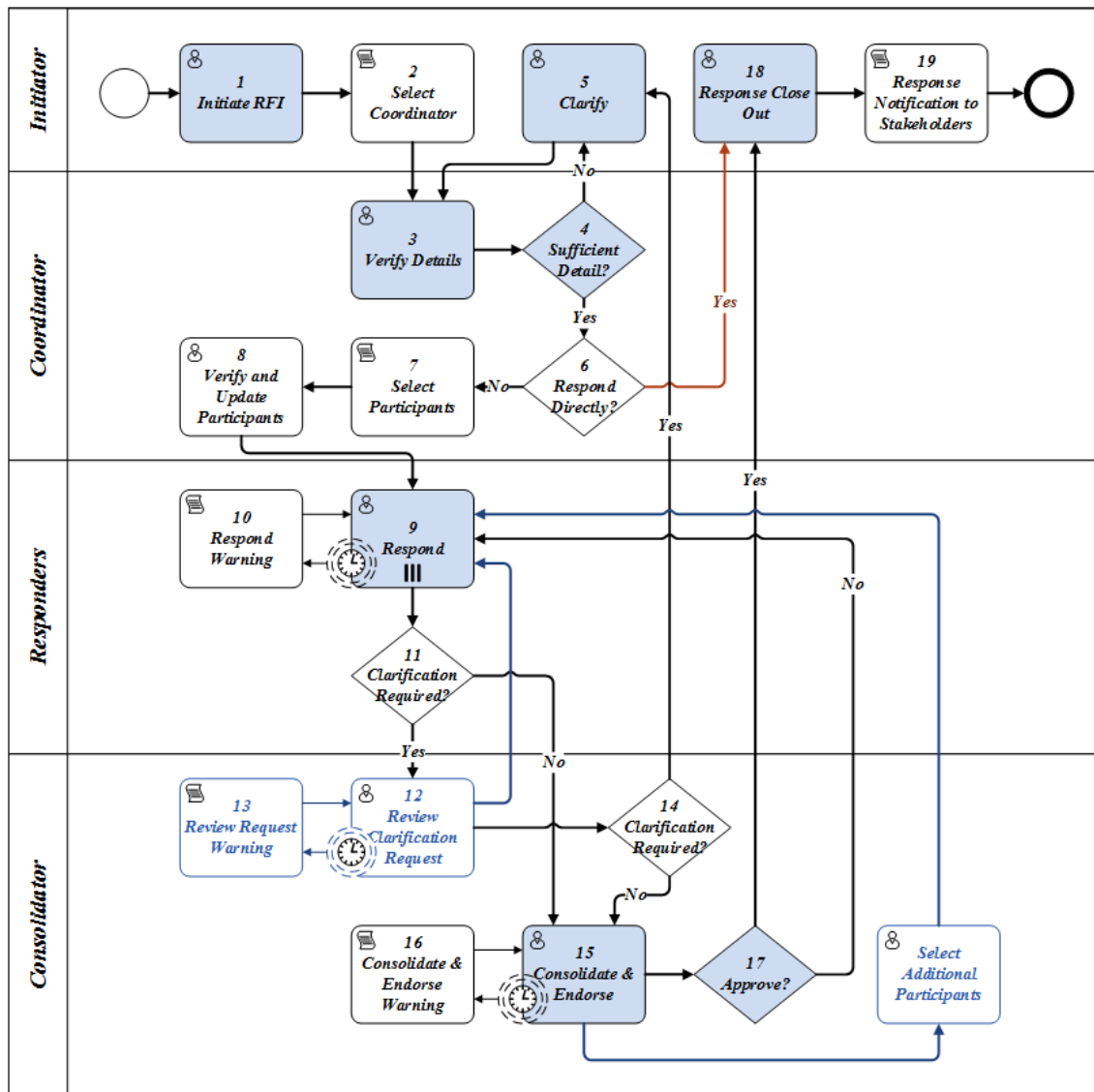


Figure 3-3 (c): High-Level Representation of the RFI Workflow Process in Project C

Figure 3-4 represents the common core structure of the RFI workflow processes shown in Figure 3-3 (a), (b), and (c). This common core structure is considered the high level structure of the RFI process and is used as a basis for deriving the foundation level RFI process. As illustrated in Figure 3-4, this workflow includes 18 steps which are performed by four different roles: *Initiator*, *Coordinator*, *Responders*, and *Consolidator*. In addition, process *Stakeholders* which are the people not actively involved of performing steps of the process, but receive communication regarding the steps, milestones, and the final result can also be considered as a role.

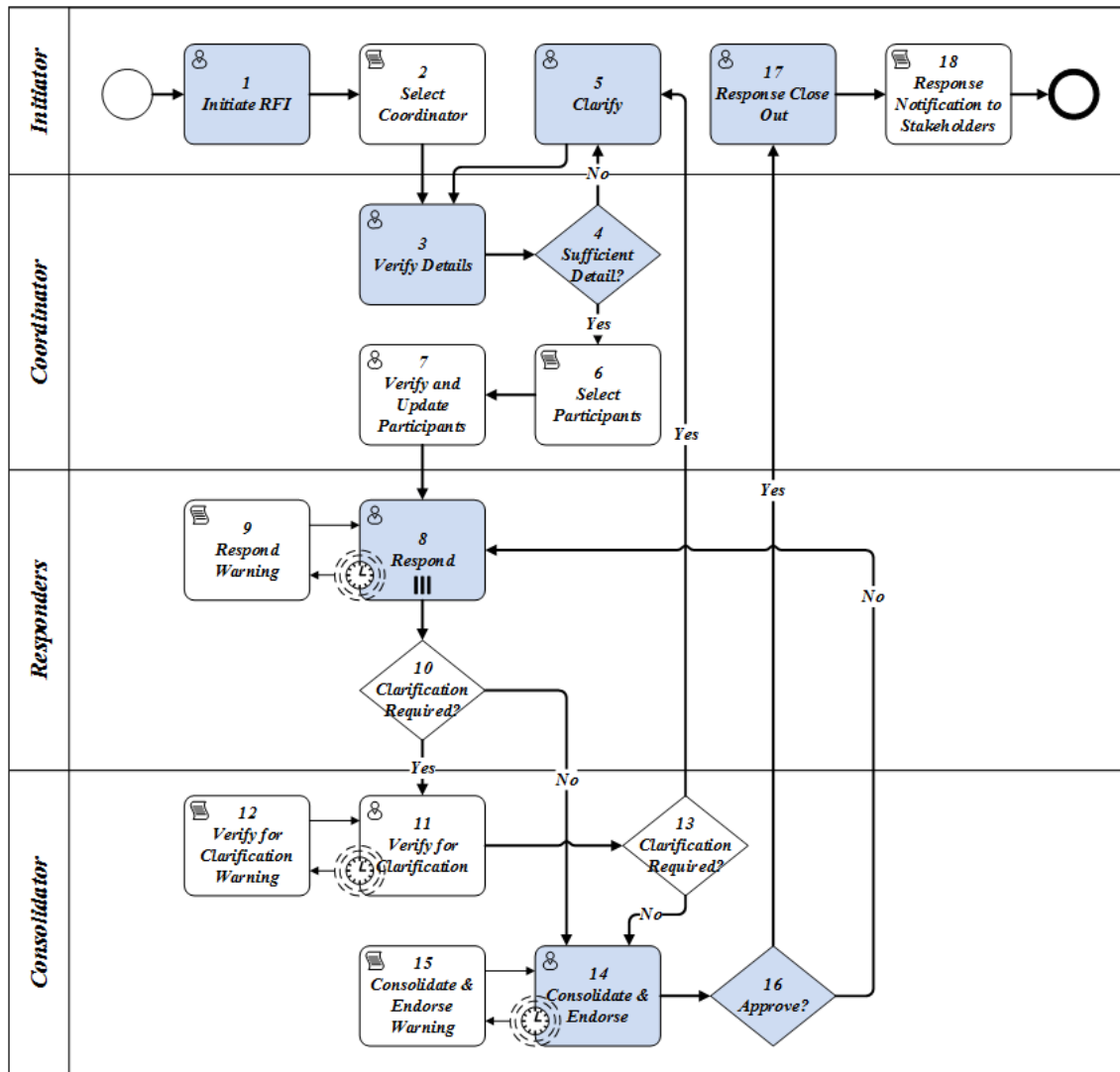


Figure 3-4: The Common Core Structure of the RFI Workflow Process

The RFI work process follows the order of activities described below:

1. Initiate – The Initiator completes the RFI electronic form and submits the detailed question and associated data, impact statement and attachments to the system, and is responsible for further clarification if requested from Coordinator or Responder(s).

The initiator can be anyone on the project team. This role can submit an RFI on behalf of another party, such as a customer or contractor.

2. **Verify** – The Coordinator triages all the requests by ensuring the details are complete, reviewing and modifying the assigned list of participants as necessary, and confirming the interval/milestone timing of the workflow.
3. **Respond** – The Responder reviews the RFI, requests clarification as necessary or composes a response and sends it on for approval.
The Responder is typically a Lead Engineer or Construction Manager, or could be another team member such as the Contract Administrator. The Responder composes and submits the response to the Approver. The Responder(s) can request clarification of any details.
4. **Consolidate and Approve** – The Approver reviews the response, and if necessary, consolidates multiple responses. The Approver also authorizes clarification requests. If the Approver deems the response insufficient, he/she returns it to the Responder. Or if the response is sufficient, the Approver issues it to the Initiator.
5. The Consolidator or Approver is the project team member responsible for consolidating, authorizing and issuing the RFI response to the Initiator. The Consolidator also authorizes clarification requests from the Responder prior to directing them to the Initiator.
6. **Close** – The Initiator receives and acknowledges the response. The RFI is closed.

3.4 Defining IFPs Based on Well-Known Best Practices

Construction projects have several operations or management activities in common. Examples include managing change, risks, contracts, procurement, and cost. Such common operations are typically associated with suggestions, recommendations, and guidelines of how to perform them more efficiently. These guidelines are generally known as best practices.

Automation of such common operations via employment of workflow processes and benefiting from their associated best practices is an appropriate approach for project performance improvement. However, due to essential differences between practices and processes adoption of best practices into workflow processes is not always straightforward.

To define a particular management activity and its associated best practices as a workflow process the following requirements should be satisfied: (1) the need for automation, such as the repetitive

or iterative nature of the operation, and the potential to increase speed, accuracy, and quality via automation, (2) process definition requirements, such as the sequence of activities, and repetitive predictable results, and (3) workflow requirements, such as participants with specific roles, and flow of work or information among the participants.

In sections 3.4.1 and 3.4.2, two frameworks are proposed for transformation of well-known best practices associated with common construction operations into workflow processes which are suitable for implementation via workflow management systems. The frameworks describe how the components of a best practice can be associated with elements of a structured process. Moreover, the frameworks explain how the inherent knowledge of best practices can be combined with the key characteristics of structured processes, such as well-defined steps, sequence, and execution constraints. The outcome is processes with the essence of best practices that can be embedded into and automated through workflow management systems.

3.4.1 Abstract Framework

A practice as a form of knowledge includes different types of knowledge: *explicit*, *tacit*, and *implicit* (Anand & Singh, 2011; Faust, 2007). *Explicit* knowledge is the category of knowledge that can easily be identified, codified, stored, and retrieved. It can easily be articulated or written down, such as rules and facts in an organization. *Tacit* knowledge is inherent with the skills and experience of people, and is hard to capture and codify, such as the skills and experience of employees of how to perform a task effectively. Part of the tacit knowledge that is difficult to reveal, but still possible to capture by observation or training is called *implicit* knowledge.

Knowledge management models explain that the explicit knowledge can be transferred more easily. Implicit knowledge needs careful observation and attention to details of how an expert is doing the work to reveal the knowledge and make it explicit, before it can be transferred. The tacit knowledge represents the mental model of the actor for performing the work, and it is transferred to somebody else only by apprenticeship, training, and experience (Faust, 2007). Well-known best practices are valuable for the explicit and implicit knowledge that they include, as well as guidelines and recommendations for how to perform activities that require tacit knowledge, but they cannot substitute the need for skillful experts that perform the work with their tacit knowledge.

The key approaches of transferring the tacit knowledge of a best practice is face to face interaction, such as meetings, workshops, coaching, and training.

Accordingly, to define a practice as a process, knowledge components of a best practice can be associated with the elements of a process in the following classification. (1) The structure of the process defines what is performed with clear steps, and it is associated with the explicit knowledge presented by the practice. (2) The human-tasks of a process are the activities that require the expert skills, experience, and judgement and cannot be automated. These tasks are associated with the tacit knowledge of the best practice and might include suggestion or guidelines for how to perform the task, but only an expert can perform the task efficiently. (3) The behavior of the process is associated with the implicit knowledge of the practice. A well-defined and efficient process is the result of implicit knowledge that is hard to capture, but can be captured by attention to details and observing the behavior and improving the process over time. (Table 3-1) represents this framework.

Table 3-1: Types of Knowledge in a Practice and their Association with Process Elements

PRACTICE COMPONENTS	... ASSOCIATION WITH ...	PROCESS ELEMENTS
Explicit Knowledge	... What is done ...	Structure of the Process
Tacit Knowledge	... Who accomplish ...	Human-tasks of the Process
Implicit Knowledge	... How is defined ...	Behavior of the Process

3.4.2 Pragmatic Framework

The knowledge inherent in best practices typically includes strategic guidelines of what to do, and tactical suggestions of how to do an operation to achieve an improved or desired outcome. This knowledge is more general and needs to be operationalized via workflow processes. A process might support strategic, tactical, and operational decisions, yet the process implementation via workflow management systems requires operational details. A process map can describe a process in different levels of abstraction, with the appropriate amount of detail (IIBA, 2015): a higher level abstraction of a process describing what is being performed and a lower level representation of how it is done with operational details, such as roles and responsibilities of the actors, and implementation details.

To define a process with the essence of its associated best practices, the following pragmatic framework is proposed: (1) classify the main components of the practice and describe their logical

relationship and define their order of execution as one or more high-level organizational processes, (2) identify process stakeholders, define the roles and responsibilities of the actors, and add the required implementation level details, to transform organizational processes into well-defined structured processes implementable via workflow management systems, and (3) define foundation processes by keeping only the core structure, the essential features, and required properties. These steps are presented in Figure 3-5 and are discussed with examples in the following sections.

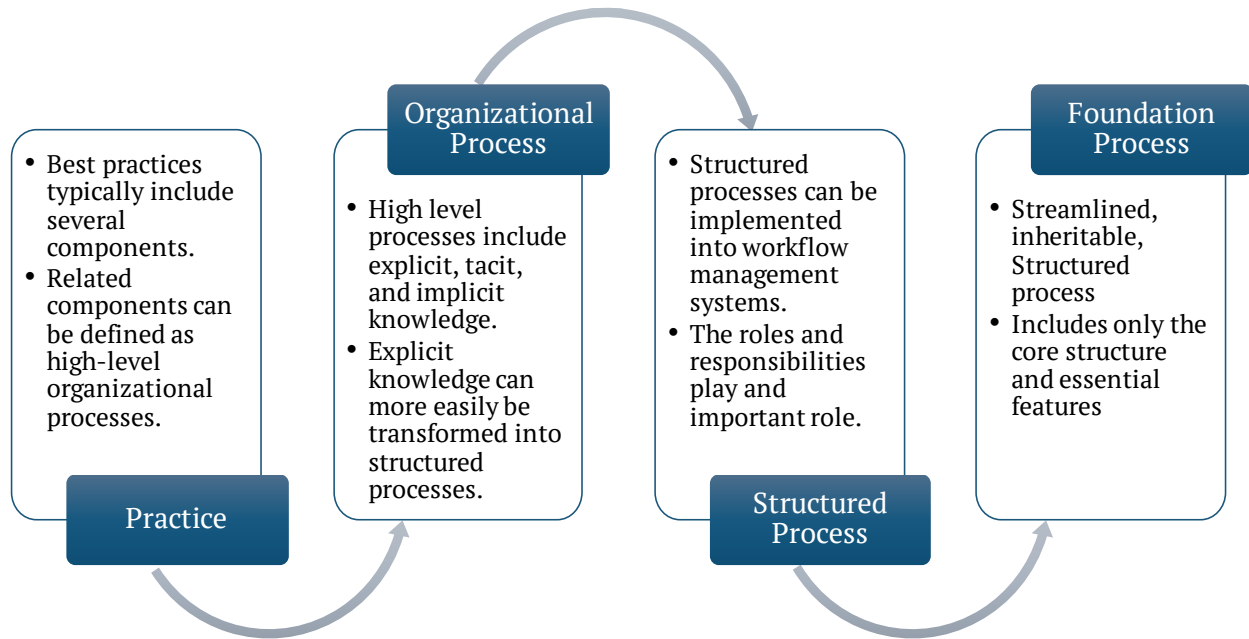


Figure 3-5: Transforming a Practice into a Structured Process

Exploring some of the well-known best practices in the domain of the construction industry, such as change management, materials management, work packaging, modularization, and lessons learned confirms that the guidelines suggested by the best practices either include some high-level processes or they can be defined as high-level organizational processes. As an example, CII best practice publication for change management offers five principles each of which has been defined as an organizational process (*Project Change Management - Special Publication 43-1*, 1994). Figure 3-6 illustrates the five principles for change management offered by CII change management best practice.

An organizational process is a high level process that includes the conceptual steps of performing work, but does not include all the details of the steps, and the execution constraints that are necessary for implementation of the process. Organizational processes cannot directly be implemented into workflow management systems. For instance,

Table 3-2 presents such a high-level process offered by CII change management best practice for the “Evaluate Change” principle (*Project Change Management - Special Publication 43-1*, 1994). As it is evident, this process cannot be implemented in a workflow management system in its current form, and lacks the required structure and details. Organizational processes should be transformed into operational structured processes for implementation through workflow management systems.

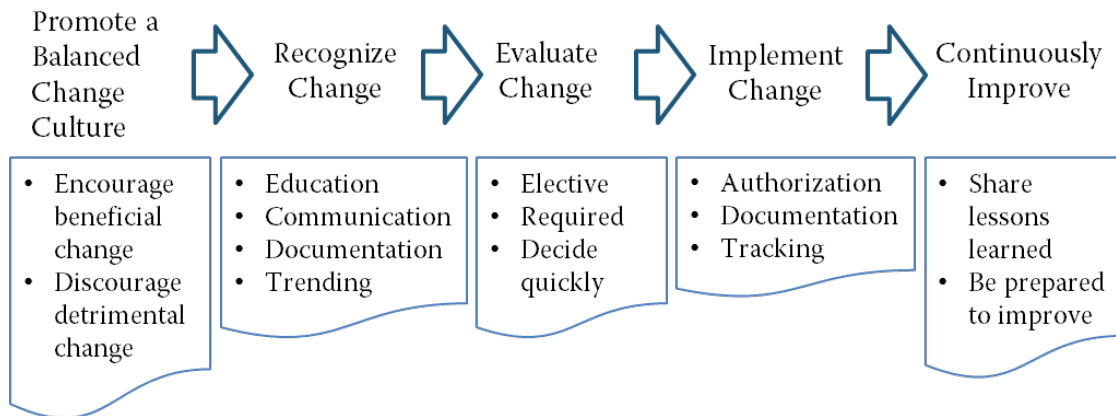


Figure 3-6: CII Change Management Principles, Each Offered as an Organizational Process

Table 3-2: Evaluate Change Process

3.1 Determine the time frame for change decision.
3.1.1 Immediate or high priority decision required? If not, process through routine measures.
3.1.2 Determine funding source for handling interim approval of a high priority change decision.
3.2 Collect data needed.
3.2.1 Conduct a thorough analysis on cost, schedule, quality, safety, resources, and other items. Evaluate on both direct and associated indirect costs.
3.2.2 Propose and evaluate alternate solutions and options.
3.3 Identify impacts.
3.3.1 Finalize impact on cost and schedule.
3.3.1.1 Primary impacts.
3.3.1.2 Secondary (indirect/ripple/cumulative) impacts.
3.3.2 Route to all involved disciplines/functions/organizations for impact.
3.4 Determine final funding source or “who pays” (cost reimbursable, design development, lump sum, and others). If applicable, confirm the interim funding source decision.
3.5 Re-evaluate project feasibility with proposed change included.
3.5.1 If change makes project unfeasible, determine whether it is a required or an elective change.
3.6 Authorize change and send out notice to all affected organizations/disciplines.

To define an organizational process as a structured workflow process two key characteristics of such a process should be considered: (1) the flow of work or information among participants with clearly defined roles and responsibilities, and (2) the structured definition of the process with the required implementation level details.

A key characteristic of a workflow process is the flow of work or information among participants. Therefore, the steps of the process should be defined as activities that are performed by the participants while considering the flow of work or information. As such, the role and the responsibility of the participants should be clarified in a workflow process. The Responsibility Assignment Matrix (RACI chart) is the proper tool for this purpose. RACI is an acronym that stands for Responsible, Accountable, Consulted, and Informed.

Moreover, the workflow should be defined as a structured process. A process in which the sequence of activities and their execution constraints are completely defined is called a structured process. For example, a change request (CR) workflow process is a formal process frequently used for authorizing any change in the scope, cost, or schedule of a project. Figure 3-7 represents the main steps of the CR workflow process and their sequence, but this is not a structured representation of the CR process.



Figure 3-7: Main Steps of a Change Request (CR) Workflow Process

Formal process modeling tools and techniques, such as Unified Modeling Language (UML), Business Process Execution Language (BPEL), or Business Process Modeling and Notation (BPMN), are being used to map a structured process. Such standard notations are required for the automation of structured processes via workflow management systems. Figure 3-8 presents the change request workflow in BPMN notation.

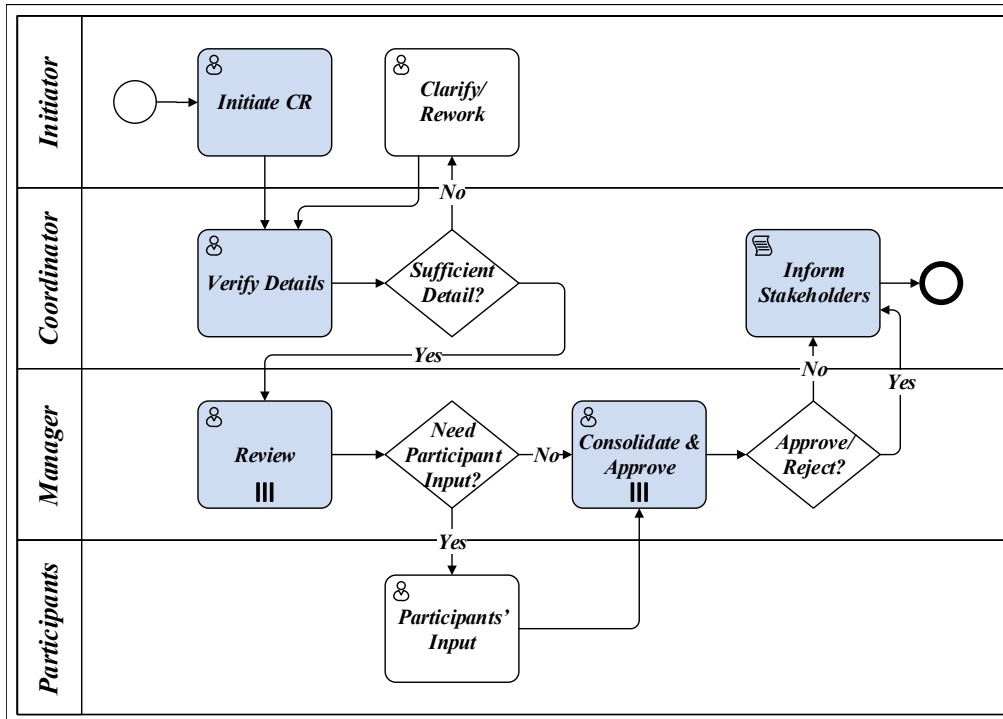


Figure 3-8: A Change Request (CR) Process in BPMN Notation

3.5 Discussion

In this chapter, the IFP modeling system was introduced and the development approaches for its processes were discussed. Two frameworks – an abstract and a pragmatic – were proposed for adoption of best practices through integration with structured processes implementable into workflow management systems. The proposed frameworks suggest that specific elements of best practices can more easily be transformed into structured processes. The end result would be a structured process with the essence of best practices that can be implemented and automated via workflow management systems.

Integration of best practices into workflow processes facilitates more consistent and more scalable adoption of best practices; however, due to fundamental differences between practices and processes there are limitations associated with the application of this approach:

1. Workflow processes facilitate the flow of work or information among participants, and have a specific structure with particular components, such as automated and human-performed activities, the sequence and logical relationship among activities, the flow of work or information, and participants with specific roles and responsibilities. Therefore,

not every element or detail suggested by a best practice can be incorporated into a workflow process. A workflow process that is defined based on best practices, thus might include an essence of the best practices, but it would not be in any sense a complete replacement for the practice.

2. Workflow processes that are based on best practices facilitate automation of particular activities, based on the recommendations of best practices. Such workflow processes include automated and human-performed activities. Human tasks should be performed by experts who are well informed of their roles and responsibilities. In other words, the workflow process would not be a replacement for the required skills, knowledge, and experience of the actors who perform those activities.

The required features and the essential properties of IFP processes are discussed in the next chapter as components of the IFP ontology.

Chapter 4

Proposed IFP Ontology

4.1 IFP Ontology

Based on synthesis of the literature, examination of functional and operational requirements for IFP system, and consultation with industry experts, this research proposes an ontology for the IFP system with the following eight components (Figure 4-1): (1) a versioning system and an applicable scope, (2) a core structure and functionality, (3) defined abstraction level to essential details, (4) associated data structures, (5) suggested practices, (6) workflow inheritance property, (7) process conformance, and (8) interoperability with other workflow processes.

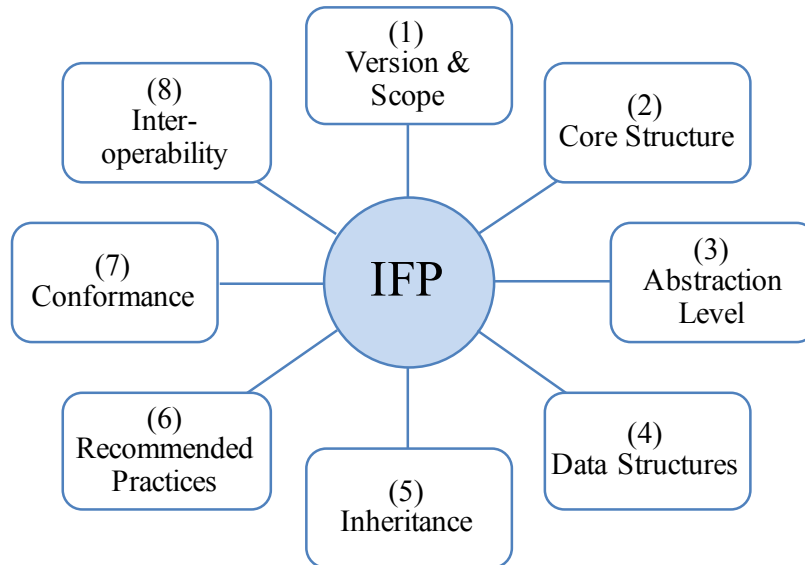


Figure 4-1: Proposed IFP Ontology

This chapter is an overview of the ontology components and discusses each one in more detail. The last two components – process conformance and interoperability – are among the direct benefits of using the IFP system, and can be considered as outcomes rather than components. However, process conformance and interoperability affect the definition of other components, such as the core structure, data structures, and the abstraction level, and thus they should be considered as components of the ontology while developing an IFP process.

4.2 Version and Scope

In any organization, processes evolve over time and process improvements and updates are supported by a process improvement framework and a versioning system. To allow future updates, a versioning system should be considered during the development of the IFP processes. Version numbers for each IFP process represent their improvements and updates.

The domain of this work, and thus the domain of the Industry Foundation Processes (IFP) system is the Architecture, Engineering, Construction, and Facilities Management (AEC/FM) industry which is also the domain of the Industry Foundation Classes (IFC). However, the concept of the IFP system and the development methodology can be adopted by any other industry.

IFP processes are developed in their most generic form, either for the AEC/FM domain, or for specific types of projects, and their scope is defined accordingly. For example, the scope of a more general process like request for information (RFI) that is used similarly in every type of project is defined as AEC/FM, and the scope of a more specific process like interface management (IM) which is useful only in large industrial projects is defined as large-scale industrial projects. As such, sets of IFP processes can be defined for a particular project types, such as oil and gas, industrial, commercial, or infrastructure. Thus, a change management IFP process developed for large-scale oil and gas projects might have a different scope comparing with a change management IFP process defined for a smaller-scale commercial project. In addition to the project type, projects delivery method and size can also affect the scope of IFP processes. Later, any IFP process can be customized more to suit any specific project.

4.3 Core Structure

Any process has a core structure that includes essential activities and their relationships. Selecting a complex process, and repeatedly substituting its activities and relationships with more abstract ones, results in a set of activities and relationships that are elemental, but sufficient for representing the purpose of that process (Malone, Crowston, & Herman, 2003). Additional activities and relationships are typically added to the core structure to customize the process for specific purposes or conditions, but if any of the essential activities and relationships removed, the meaning of the process might not be preserved. Extracting the minimal yet essential elements of a complex

process, developed based on the industry best practices and improved incrementally through the process improvement cycle, results in the core structure required for defining an IFP process. For example, as outlined previously, extracting the core structure of implementation-level RFI processes, such as the process shown in Figure 3-4, results in the minimal yet essential activities and relationships presented as a simple structured process in Figure 4-2.

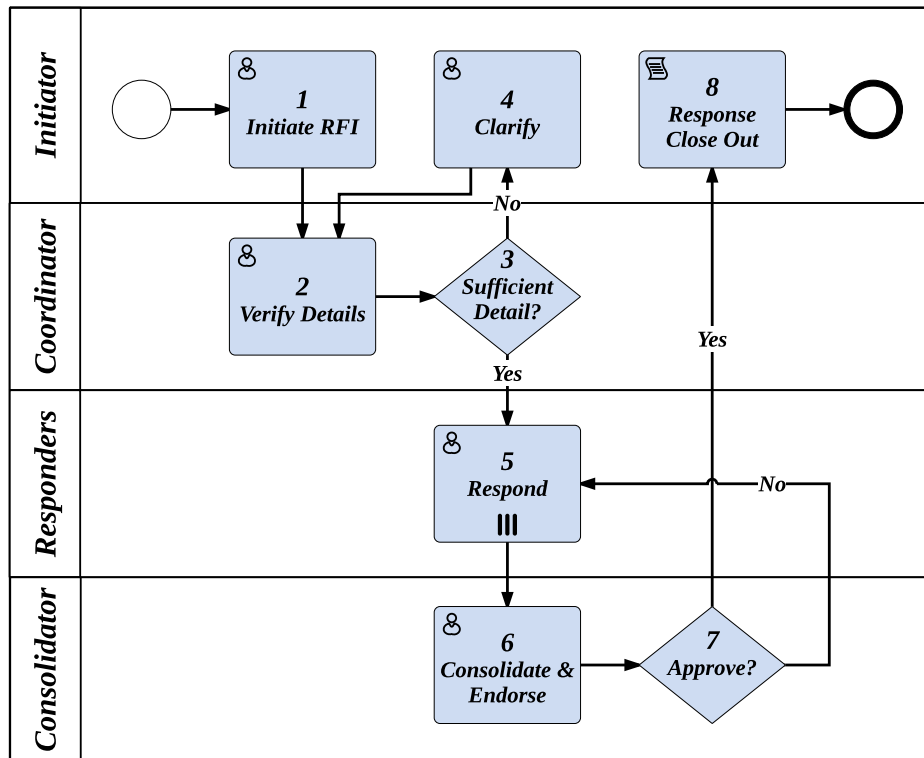


Figure 4-2: The Core Structure of an IFP for the RFI Process

As such, the core structure of an RFI process includes the following steps. (1) A project team member initiates a request. (2) A coordinator verifies the request for accuracy and completeness, and assigns/confirm participants. (3) If any clarification is necessary, (4) the request is being sent to the initiator for clarification; if not, (5) it is being sent to one or more participants, typically a lead engineer or a construction manager, for composing a response. (6) The consolidator is then responsible for consolidating responses, (7) and approving and issuing the response to initiator; (8) and finally all process stakeholders are informed and the workflow is closed.

4.4 Abstraction Level

The abstraction level of a workflow process is important because it determines the amount of detail that the process is represented with. A process can be characterized in a high-level abstraction level that explains the process steps, or it can be defined as a structured process in which the sequence of activities and their execution constraints are completely defined.

Furthermore, a process can be presented with operational details that include activities and their relationships, or it can be defined with implementation details that contain information on execution and technical details required for enactment of the process in a computerized system.

Table 4-1: Workflow Abstraction Levels

ABSTRACTION LEVEL	DESCRIPTION
Meta-level Workflow	A conceptual description of the process flow Includes organizational-level details
Foundation-level Workflow	A high-level structured definition with particular properties Includes operational-level details
Workflow Template	A customized workflow with the most common components Includes operational-level details
Workflow Implementation	An implemented workflow for a specific organization or project Includes implementation-level details
Workflow Instance	An executed instance of an implemented workflow Includes implementation-level details

Based on an examination of industry practices, an analysis of the literature, and the required level of details for the IFP system, workflow processes are classified into the following five abstraction levels (Table 4-1): (1) meta-workflows, (2) foundation-level workflows, (3) workflow templates, (4) workflow implementations, and (5) workflow instances. Moreover, the foundation-level is proposed as the appropriate level of abstraction for IFP processes.

A meta-workflow is a conceptual definition of a workflow, either textual or in a flow-chart format. It is not a structured definition of a workflow and its main purpose is to describe the workflow behavior. A foundation-level workflow, associated with the concept of Industry Foundation Processes, however, is a structured definition of a process, with some operational and implementation level details that are required for its proper functioning. It is the highest abstraction level implementable in a workflow engine enabled environment, such as Skelta or Microsoft Workflow Foundation which are the environments used in this research.

A workflow template is a customized workflow, based on an IFP, that contains the most common activities and relationships for a particular type of project. It can be used as the starting point for deriving more detailed implementation-level workflows suitable for a specific project. Workflow implementations typically include all the required human-oriented tasks, as well as automated tasks, such as writing to databases and sending notifications to participants, as required.

An executed version of an implementation-level workflow is called a workflow instance. For any implementation level workflow, several workflow instances are typically created throughout the lifecycle of the project. Some workflow instances might have a relatively short lifetime, and some might be active for a longer period of time, before completing their execution and closing out. Each workflow instance typically stores all the data associated with its execution steps. For example, the execution details of an activity called "Verify Details", which is one step within the RFI workflow, include details such as, instance identification code, accessed time and date, completed time and date, name of responsible and responding party, current status of workflow, and more. All workflow instance execution data are stored in databases for retrieval and analysis, for auditing purposes, or to improve the definition of the workflow.

As an example, an IFP process for deliverables management with the domain of AEC/FM and the scope of industrial projects, can be customized to a deliverables management workflow template suited for oil and gas projects, and then customized and implemented for a specific project, with several instances of the workflow running simultaneously on a workflow management system.

4.5 Data Structures

Processes rely on particular data structures for their proper functioning throughout the execution steps. A process stores, manipulates, and passes information with the flow of work from one step to another. For example, the execution of a request for information (RFI) process requires data fields, such as RFI ID, Contract ID, Title, Description, Request Date, Response Date, etc. Some data structure fields are being manipulated within the subsequent execution steps, such as "Response Note", and some of them even determine the flow of work while executing the process. For example, the flow of work might be redirected to a different person depending on the time or cost impact of the request.

Table 4-2: Minimal Set of Data Structure Fields for an RFI Process

RFI ID	Title	Request Reason
Contract ID	Description	Need Date
Project ID	Unit	Responder
Request Type	Area	Response Note
Requested By	Discipline	Response Date
Request Date	System	Coordinator
Cost Impact	Status	Approve Date
Schedule Impact	Priority	Final Response

An IFP is defined with a minimal set of data structures that are required for its proper implementation. Table 4-2 presents a minimal data set that is associated with an RFI process. Some of the data and metadata fields are automatically assigned by the workflow management system, *e.g. Process ID, Response Date, and Approve Date*, and some of them are entered by process participants in each step of the process. Additional fields can be added when required, but the minimal set that is defined within an IFP is kept while customizing a process.

4.6 Recommended Practice

Recommended practices are guidelines for how to perform each of the human-tasks in an IFP process. These guidelines are not comprehensive and cannot be a substitute for the knowledge, skills, and the experience of the actor, but they are useful in identifying and performing the main steps and requirements for performing those activities.

Several recommendations and guidelines that are available in best practices can be used as guidelines for performing the human-tasks.

4.7 Inheritance

In computer science, inheritance is a key programming concept. Inheritance enables reuse of code by keeping certain properties of an object called a super-class, while transforming it into a new object called a sub-class. Sub-classes typically include extra or more detailed features, while inheriting features from the super-class. Super-classes are also called parent-classes or base-classes, which sub-classes are also called child-classes or derived-classes. The inheritance concept can be applied to the IFP system whereby the core structure and particular properties of an IFP workflow is inherited, and additional activities or properties are added to form a customized

version of the workflow. The idea of using the inheritance concept for workflow processes is not new; Van der Aalst explored the concept of workflow inheritance (W. M. van der Aalst, 2002; Van Der Aalst, 2003; W.M.P van der Aalst & Basten, 2002) and developed four types of workflow structural inheritance: protocol, projection, protocol/projection, and life-cycle inheritance. A detailed description of these workflow inheritance notions is beyond the scope of this paper, and the reader is referred to the cited references for more information.

This research offers three categories of inheritance for workflow processes to facilitate conformance with regulatory requirements or institutional practices: (1) Structural, (2) Organizational, and (3) Temporal, and defines sets of workflow inheritance rules for structural and organizational inheritance to allow or restrict certain workflow transformations. These inheritance rules control how more detailed implementation-level processes are derived from an IFP, while maintaining conformance to the IFP. Structural inheritance rules restrict the flow of work or information in subclasses of a workflow to the sequence and set of core activities defined in a superclass IFP. This ensures that the core structure of an IFP process does not change when it is customized to accommodate specific project requirements.

Organizational inheritance rules ensure that the level and sequence of authorization defined in an organization or project is met with the execution of the workflow process. For example, if someone is not available who would be the next responsible person to whom the work or information be directed, or who could be assigned as a delegate for somebody who is not available for a period of time. For this purpose, a responsibility assignment matrix, *i.e.* a RACI chart is used to define the participation of various process stakeholders with their defined roles, responsibilities, and deliverables in completing each step of the process. A sample of a RACI chart is presented in Table 4-3.

Table 4-3: Sample of a RACI Chart

ACTIVITIES	ROLE 1	ROLE 2	ROLE 3	ROLE 4	ROLE 5	START	FINISH
Activity 1	I	R	A	A	I	10-Mar	18-Jul
Activity 2	R	I	A	A	I	11-Sep	15-Dec
Activity 3	I	I	R	I	C	14-Sep	16-Nov
Activity 4	A	R	I	I	A	12-Oct	03-Dec

Temporal inheritance rules define allowable durations for each activity according to regulatory or contractual obligations or industry best practices. For example, how much time is allowed for an approval activity to be finalized according to regulatory, institutional, or contractual obligations. Table 4-4 presents a set of structural inheritance rules to preserve the presence and the sequence of core activities in a customized workflow process. In addition, it offers a sample of organizational inheritance rules. These are a sample of rules that can be used to ensure conformance with regulatory requirements or institutional practices. Organizational and temporal rules are defined as properties associated with the core structure of an IFP process, and thus the structural inheritance rules are the most important rules for conformance checking. In this research we focus on the structural inheritance rules.

Table 4-4: Sample of Workflow Inheritance Rules

CATEGORY	INHERITANCE RULES
Core Activities	Core activities should not be removed, <i>e.g. request, verify details, respond, and approve in an RFI process.</i> The sequence of core activities should not be modified (W6). A connection from an activity to any of its predecessor activities might be added (W1). One core activity may be distributed into two or more activities (W2), <i>e.g. double-stage approval.</i>
Additional Activities	Additional activities might be added between core activities (W3). Additional activities should not create a parallel path in the workflow (W7, W8). But, additional activities might bring the flow to a predecessor activity (W4). An additional activity can be in relationship to one activity (W5).
Roles & Responsibilities	Extra roles might be added. A lower-ranked role cannot approve the work of a higher-ranked role. Responsibilities of a role might be delegated to another role. Different roles might have the same responsibility level.

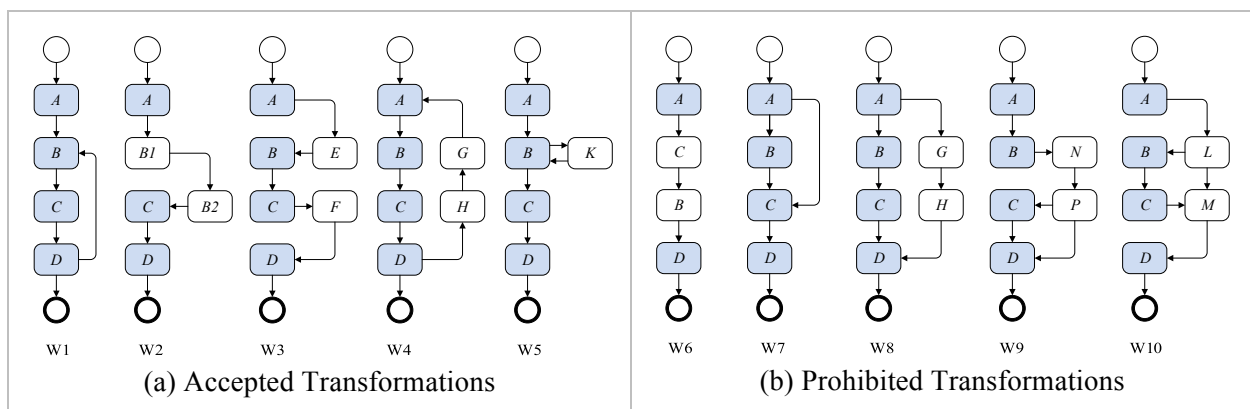


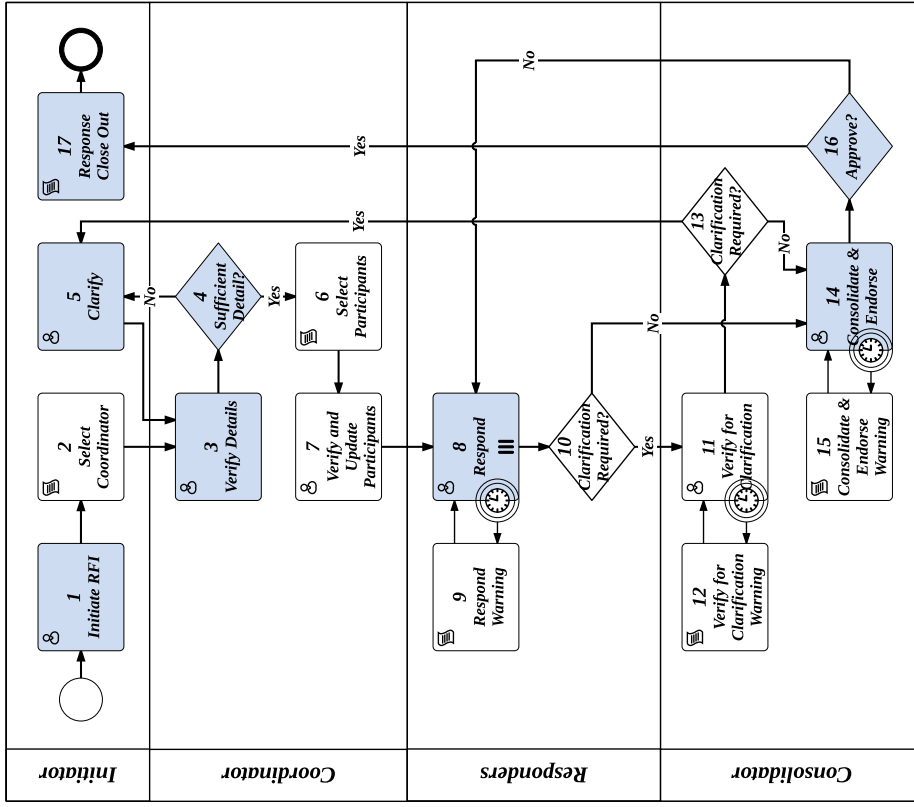
Figure 4-3: Examples of Accepted and Prohibited Transformations

Figure 4-3 graphically presents accepted and prohibited transformations for a simple specification workflow $A \rightarrow B \rightarrow C \rightarrow D$ in which the flow of work is only possible through A then B then C and then D. As demonstrated in Figure 4-3(a), it is accepted for the super-class specification workflow of $A \rightarrow B \rightarrow C \rightarrow D$ to be transformed into sub-class workflows presented as W1 through W5. In all of these transformations none of the core activities can be skipped or their sequence be altered. W2 represents dividing an activity into two, in which part of the enactment of task B is performed in task B1 by one person, and the rest is performed in B2 by someone else.

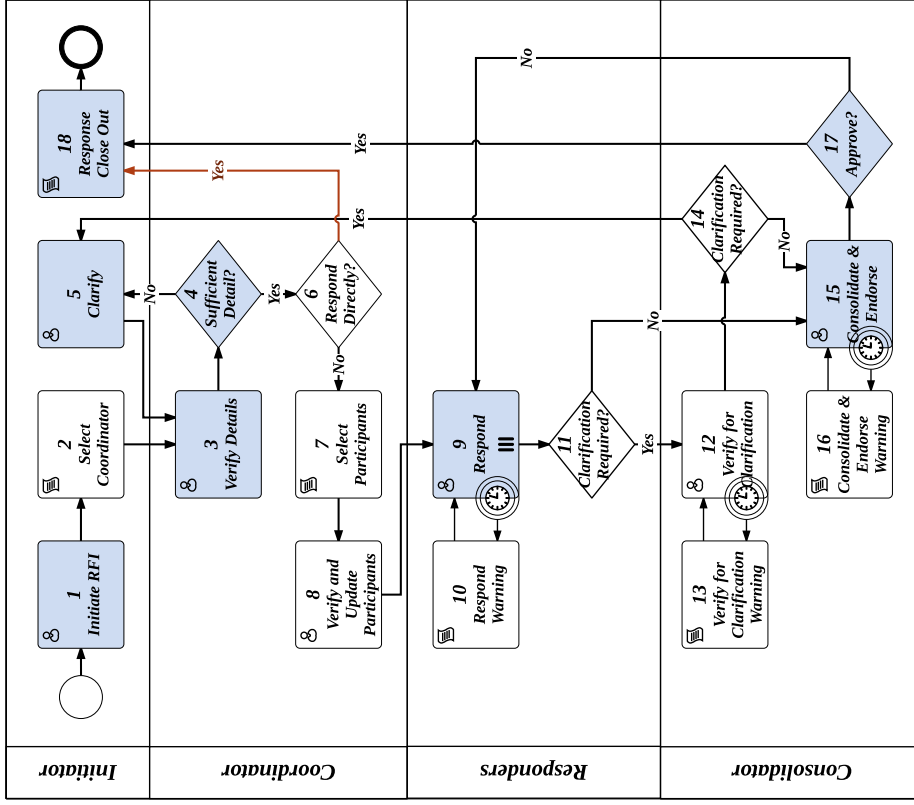
Figure 4-3(b) presents a set of transformations for the specification workflow $A \rightarrow B \rightarrow C \rightarrow D$ that are prohibited according to the defined workflow inheritance rules. Sequence of activities should not be changed (W6). Parallel paths are not allowed (W7, W8) by which the execution of some core activities might be circumvented. While new blocks of activities might be added between two adjacent existing activities, they should not be connected to any successor activities (W9, W10). For instance, W3 is an accepted transformation, but W10 is not. The inheritance rules ensure that all the core activities are present, and the sequence of their execution is not altered. Workflow inheritance is a key feature of Industry Foundation Processes. It enables reusability and customization of IFPs for different project circumstances, and is a basis for IFP conformance and interoperability.

4.8 Conformance

Conformance of customized complex processes to their associated IFP process facilitates transparency, and streamlines process improvement and reengineering. IFP inheritance as a key property of the IFP system provides a method for systematic evolution of IFP processes into more complex customized implementations for a specific project, while maintaining conformance to requirements. Enforcing the inheritance rules at the workflow design stage ensures that sub-classes of a particular workflow are in conformance with its associated IFP. This is called forward conformance checking. Conversely, a workflow process can be designed with no structural restrictions at the design stage. In this case the customized version of a workflow can then be compared with its associated IFP according to the inheritance rules, to discover whether it is in conformance or not. This is called backward conformance checking.



(a) An RFI Process in Conformance with the IFP



(b) An RFI Process Not in Conformance with the IFP

Figure 4-4: Conformant and Non-Conformant Versions of the RFI Process

For example, Figure 4-4(a) demonstrates a customized version of the RFI process which is in conformance with the RFI-IFP process presented in Figure 4-2. However, based on the defined workflow inheritance rules the workflow demonstrated in Figure 4-4(b) is not in conformance with the RFI-IFP process, because of the direct connection between Activity 6 and Activity 18 which creates a parallel path. In Figure 4-4, all the core activities that are associated with the set of activities available in the IFP process are outlined in gray. The additional activities are outlined in white. The backward conformance checking is not an easy task for complex implemented versions of processes, and thus cannot effectively be guaranteed. In this paper, we present a practical solution for automated backward conformance checking of workflow processes using a first-order logic language.

4.9 Interoperability

Process interoperability is the interaction and exchange of information between cross-organizational workflow processes, and is a vital component of alignment between collaborating organizations. Process interoperability is the highest level of interoperability. It is dependent upon achieving lower levels of interoperability, such as technical, and information interoperability.

The IFP system facilitates interoperability between processes via an external view for each process that is abstracted to the IFP core structure. Any workflow process that is in conformance with the IFP includes all the core activities, and adhere to the core structure of the IFP. In IFP interoperability model, the external or public view of processes that are in conformance with the IFP is abstracted to the core structure of the IFP.

The common data structures of the IFP processes enable the essential data exchange between processes; and the structural, organizational, and temporal inheritance rules facilitate the communication of process and organizational details. The interoperability property of the IFP system, thus, relies on several other properties to facilitate interaction between workflow processes: the abstraction level, core structure, data structures, inheritance, and conformance.

Chapter 5

IFP System Validation

5.1 Validation Methodologies

To validate the functionality and benefits of the IFP process modeling system it would be ideal to use the IFP system for a set of IFP processes in one or more construction projects. However, such full deployment of the IFP system to fulfill the requirements of a real construction project is a complex task and beyond the scope of this research. In addition, application of such a system in an existing project as a case study requires several types of permits and numerous resources, which would not be feasible as part of this research.

Alternative validation methods were carefully investigated, and the following four methodologies were proposed for this research: (1) expert feedback for the proof of concept, development methodologies, and justification of the value and benefits; (2) functional demonstration of the functionality and benefits of the IFP system; (3) discrete event simulation of existing and IFP workflow processes and comparison and analysis of the results; and (4) conducting surveys from industry experts within the construction and the information technology fields.

Expert feedback is one of the main validation methodologies in each stage of the research for development, deployment, and demonstration of benefits, such as process conformance and interoperability. Process analysts, software architects, IT specialist, computer science experts, and construction management professionals have been involved in and provided expert feedback and advice during different stages of the research.

Functional demonstrations is an essential validation approach for the IFP system and is performed with the following purposes: (1) to illustrate the functionality of the IFP system through implementation of the request for information IFP process into a workflow management system, as well as deployment of a customized version of the same process, which is customized based on the workflow inheritance rules defined in Chapter 4; and (2) to demonstrate the benefits of the IFP system by developing an automated workflow conformance checking tool that uses the IFP system workflow inheritance rules to automatically check the conformance of two workflows. The

conformance checking, which will be discussed in Chapter 6, not only includes an algorithm developed with a first-order logic language for analyzing and comparing the structure of workflow processes, and a visualizer to graphically display the result of the analysis. Discrete event modeling and simulation of workflow processes is also an investigated validation methodology and is discussed in further detail in the next section of this chapter. Surveys were considered but not conducted in this research. The reasons are explained in the discussion section of this chapter.

5.2 IFP System Deployment

Deployment of the IFP system includes implementation of IFP processes and customization of them for particular projects. This section clarifies how workflow inheritance rules and program inheritance rules are used to derive a customized workflow process from an IFP workflow process, and examines implementation of an RFI process into a workflow management system.

Modification of a workflow process can be performed in three possible manners: (1) modifying the functionality of existing activities; (2) adding new activities or removing existing ones; and (3) changing the order of, or the relationship between, activities. To derive a more detailed customized workflow from an IFP modifications should be performed in a controlled manner, and the customization flexibility should be limited to preserve the conformance of the customized process with the IFP. To accomplish this, the programming inheritance concept, which is part of object-oriented programming languages, as well as the workflow inheritance methodology, which is defined through the workflow inheritance rules, are required.

Using object oriented programming languages, each *activity* and each *form* (i.e. window/screen) of a workflow process is defined as a *class*. A class is a tool used in programming to encapsulate (i.e. combine) related fields and functionality. The methods associated with each class define the functionality and behavior of its corresponding activity. In object-oriented programming *inheritance* is a concept and tool that allows one class to ‘inherit’ certain fields and functionality from an existing class; which fields and functionality are inherited is controlled by the developer. This allows similar classes to re-use code and allows more complex or customized classes to extend existing functionality. The existing class is a parent class or superclass, and the extended class is a child class or subclass. The functionality of parent class can be overwritten by the child class at the discretion of the parent class’ developer.

In the customization process, the programming inheritance concept is used whenever the behavior of an existing activity needs to be extended by adding new functionality or behavior for that activity. The extended activity is a subclass of the existing activity, inheriting the functionality and the behavior of the superclass and adding supplementary functionality. The customized workflow process would be in conformance with the IFP process as long as overriding the superclass methods is restricted to the workflow inheritance rules. For example, in Figure 4-4(a) the activity 9 is an automated notification to the activity 8. Therefore, this new functionality can be implemented into the customized process by adding a notification method to the subclass of the activity 8, using programming inheritance available in the programming language.

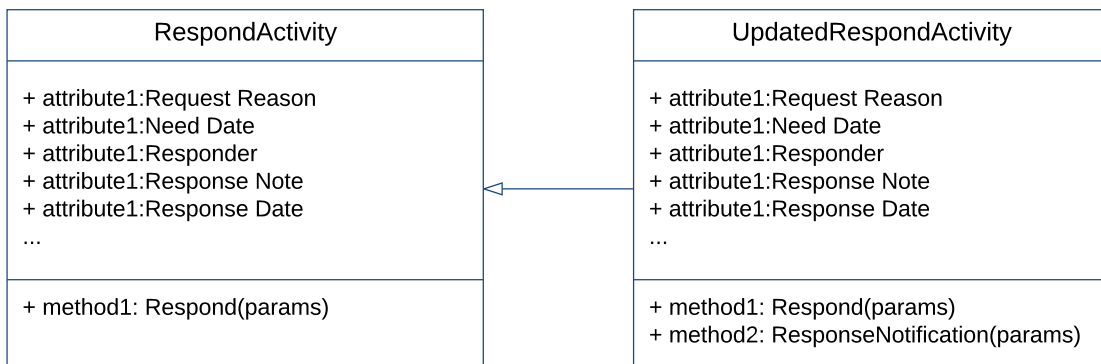


Figure 5-1: Example of Programming Inheritance for Respond Activity

However, in many cases the new functionality cannot be consolidated into its predecessor or successor activity, *e.g.* when the new activity is performed by a different role, or when the nature of work performed by the new activity is different. For example, activity 7 in Figure 4-4(a) cannot be merged into activity 8, because they are being performed by different roles (people). In such cases, the programming language inheritance is not sufficient; the workflow inheritance rules govern the customization process.

5.2.1 Deciding on the Deployment Platform

Several open-source and commercial workflow management systems are available, such as: Activiti, IBM BPM, SAP Business Workflow, Skelta BPM, Oracle BPM Suite, and Windows Workflow Foundation. Although all of them include workflow engines to enact workflow processes; their features, capabilities, and targeted domains are quite different. The focus might be

document management, collaboration management, content management, customer relationship management, or enterprise integration.

Some workflow management systems, like Activiti, are full featured business process management (BPM) suites that offer tools for the entire business process lifecycle, including design, modeling, execution, monitoring, and optimization. Others, like Windows Workflow Foundation, focus on the capabilities offered by the workflow engine as a framework for developers to expand on and build applications on top of.

For this research Windows Workflow Foundation (WF) was selected as the deployment platform, for its: flexibility, availability as part of the Visual Studio, and its suitability for modeling and enactment of long-running workflow processes. Windows Workflow Foundation 4.5 also offered improved functionality over previous versions.

5.2.2 Workflow Foundation (WF) Technology

WF technology is a component of the .NET Framework in Microsoft Visual Studio. WF offers a declarative programming environment in which the code is separated into programming fragments called activities; these activities are used to control the functionality at each stage of the workflow.

WF, in the .NET Framework 4.5, offers three control flow structures: sequence, flowchart, and state-machine. The sequence workflow model defines the flow of program as a sequence of activities. The flowchart contains flow control elements and is typically used to implement non-sequential workflows. In the flowchart model, the flow of execution of activities is based on the values of variables. State-machine provides an alternative approach to model the flow of events that cannot be anticipated. This approach relies on states and transitions between states, and is suitable for modeling workflows that involve human interactions (“State Machine Workflows,” 2015; White, 2013).

A *state machine workflow* model was chosen for this research; it allows the user to create visual, graphical representations of the workflow using nodes and arrows and determines its next step based on information submitted by the users. This model driven development is especially useful for managing complex applications and large programs - to avoid losing the structure of the program in the code details. Each state will commonly have an activity associated with it; these

activities control the work performed at that state (Microsoft Developer Network, 2015b; White, 2013). The model is executed by a runtime engine. The runtime engine, or more specifically the Common Language Runtime (CLR), not only manages the memory but also provides control for asynchronous execution (execution of code in a separate thread of the CPU), and parallel execution in a distributed system (Microsoft Developer Network, 2015a). Multiple threads are useful for modeling workflow processes because they allow multiple processes or multiple instances of a process to run concurrently.

5.2.3 Implementation of RFI Workflow Process

The C# programming language along with Microsoft Windows Workflow Foundation (WF) has been used to implement three versions of the RFI workflow, an RFI-IFP workflow that is shown in Figure 4-2, and two customized more detailed RFI workflows presented in Figure 4-4. One is in conformance with the IFP workflow and one is not. For the implementation of workflows, the state-machine model is used. A graphical representation of the model for the RFI-IFP prototype is presented in Figure 5-2.

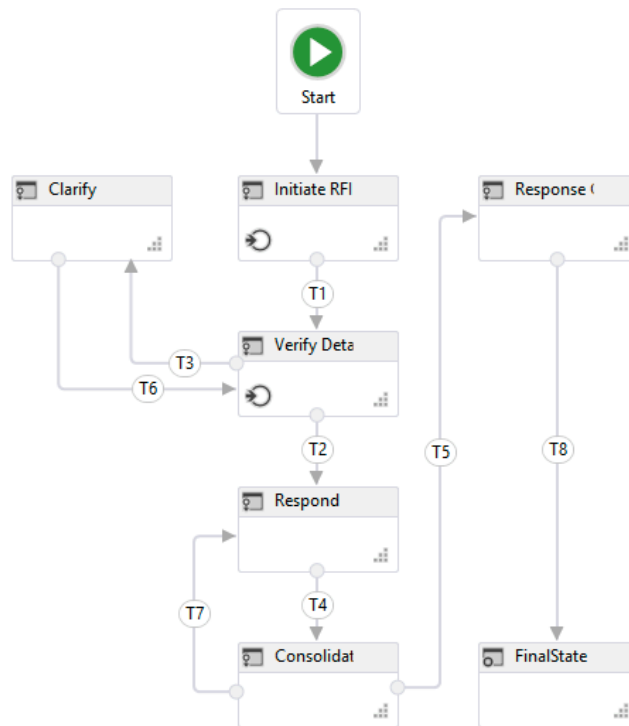
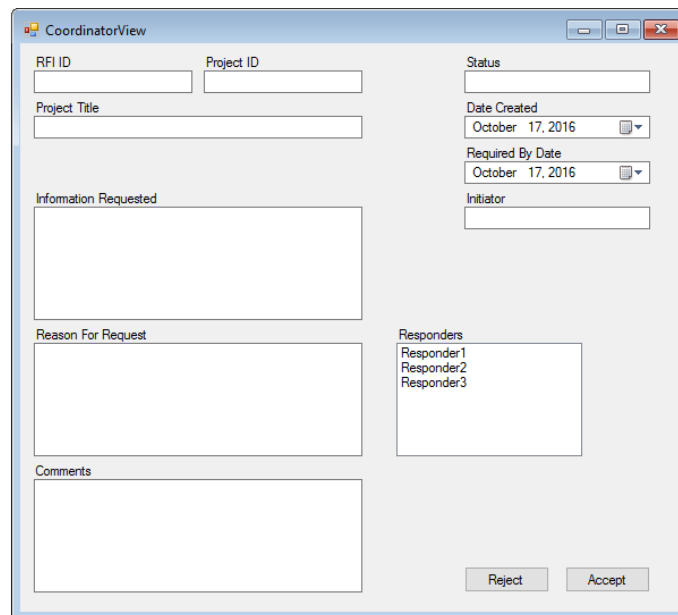


Figure 5-2: Implementation of the RFI-IFP Workflow as a State Machine Model

Construction industry workflow processes, such as the RFI process, are typically initiated over distributed systems. The flow of work or information is sent to different actors who are able to log in and perform one or more steps of the workflow. Microsoft Windows Workflow Foundation fully supports parallel and distributed computing and is a suitable platform for developing distributed systems. In WF 4.5 the process logic is defined as a workflow which is executed by the runtime engine.

A workflow process can be modeled using Workflow Foundation technology either as a web-client distributed application or as a windows-client centralized application. Web-client applications, which are used for distributed systems and accessed through a web-browser or webpage, are the most representative form of implementation for the RFI process. However, the same classes, structure and functionality that is used in a distributed system can be used in the desktop application; and the desktop application also supports multiple processes and instances running concurrently. Thus the desktop application's validation testing is also valid for distributed systems.

To reduce the complexity of RFI deployment, a windows-client desktop application has been developed using Microsoft WF and the C# programming language. The Consolidator and Coordinator views of the application are shown in Figure 5-3 and Figure 5-4 respectively.



The image shows a screenshot of a Windows application window titled "CoordinatorView". The window contains several input fields and buttons. On the left side, there are fields for "RFI ID", "Project ID", and "Project Title". Below these is a large text area labeled "Information Requested". Further down is another large text area labeled "Reason For Request". At the bottom left is a text area labeled "Comments". On the right side, there is a "Status" field, a "Date Created" dropdown menu showing "October 17, 2016", a "Required By Date" dropdown menu also showing "October 17, 2016", and an "Initiator" field. Below these is a "Responders" list containing "Responder1", "Responder2", and "Responder3". At the bottom right of the window are two buttons labeled "Reject" and "Accept".

Figure 5-3: The Coordinator View

The screenshot shows a software window titled "ConsolidatorView". It contains the following fields and controls:

- RFI ID**: Text input field.
- Project ID**: Text input field.
- Project Title**: Text input field.
- Information Requested**: Large text area.
- Reason For Request**: Large text area.
- Responder**: Text input field.
- Response**: Large text area.
- Response Comments**: Large text area.
- Consolidated Response**: Large text area.
- Status**: Text input field.
- Date Created**: Date dropdown menu (October 17, 2016).
- Required By Date**: Date dropdown menu (October 17, 2016).
- Initiator**: Text input field.
- Response Date**: Date dropdown menu (October 17, 2016).
- Approval Date**: Date dropdown menu (October 17, 2016).
- Buttons**: "Reject" and "Approve" buttons at the bottom right.

Figure 5-4: The Consolidator View

While the application is running on the system, several instances of the RFI workflow process can be enacted simultaneously. Different users can log in and complete their associated tasks. When an instance is in the state of waiting for a response from an actor, such as the responder or consolidator role, its associated information is unloaded from the computer memory to a database. When the actor who owns the task resumes the instance, process information is loaded from the database to the memory. This process is repeated in any idle time to reduce the burden on memory. The process model is saved in a XAML file – a type of XML file developed by Microsoft. This XAML file is used as an input to the automated workflow conformance checking tool, as described next.

5.3 Discrete Event Simulation (DES)

Simulation is the imitation of real-world processes or systems on a smaller scale for examination, testing, or training purposes; and is used when – due to limitations – the real system or process is not practical to study directly. Several simulation methods are available, such as discrete-event simulation, continuous simulation, system dynamics, Monte Carlo simulation, and qualitative

simulation. Discrete-event simulation and system dynamics are among the most widely-used simulation methodologies for analyzing business processes (Giaglis, 2001).

Workflow management systems document the execution details of every step in enactment of workflow processes, and are a rich source of documented events. The analysis of the process events can be performed by discrete event simulation with the focus on the behavior of completed processes, evaluating running process instances, or predicting the behavior of future process instances (Mühlen & Shapiro, 2010). Discrete event simulation is also used for process improvement purposes by detecting bottlenecks, providing visibility, identifying rarely used paths, and to offer an improved version by comparing the efficiency of the original and the updated workflows.

5.3.1 Simulation of RFI Workflow Process

Since the application of the IFP system through implementation of IFP processes in real projects is beyond the limitations and available resources of this research, discrete event simulation was selected as the viable and suitable method for modeling and analyzing those processes and comparing the behavior of the existing processes with the behavior of the IFP system processes.

Figure 5-5 shows an RFI process that has been used in an oil and gas project in Canada. The process is comprised of nine versions that have been improved over the lifecycle of the project. Each version includes several executed instances. Version numbers and the number of instances associated with each version are summarized in Table 5-1 with the total record of 22 840 instances.

Table 5-1: Versions of the RFI Workflow

VERSION	INSTANCES	VERSION	INSTANCES
Ver. 1	62	Ver. 6	2197
Ver. 2	323	Ver. 7	2326
Ver. 3	1403	Ver. 8	13233
Ver. 4	404	Ver. 9	2805
Ver. 5	87	TOTAL	22840

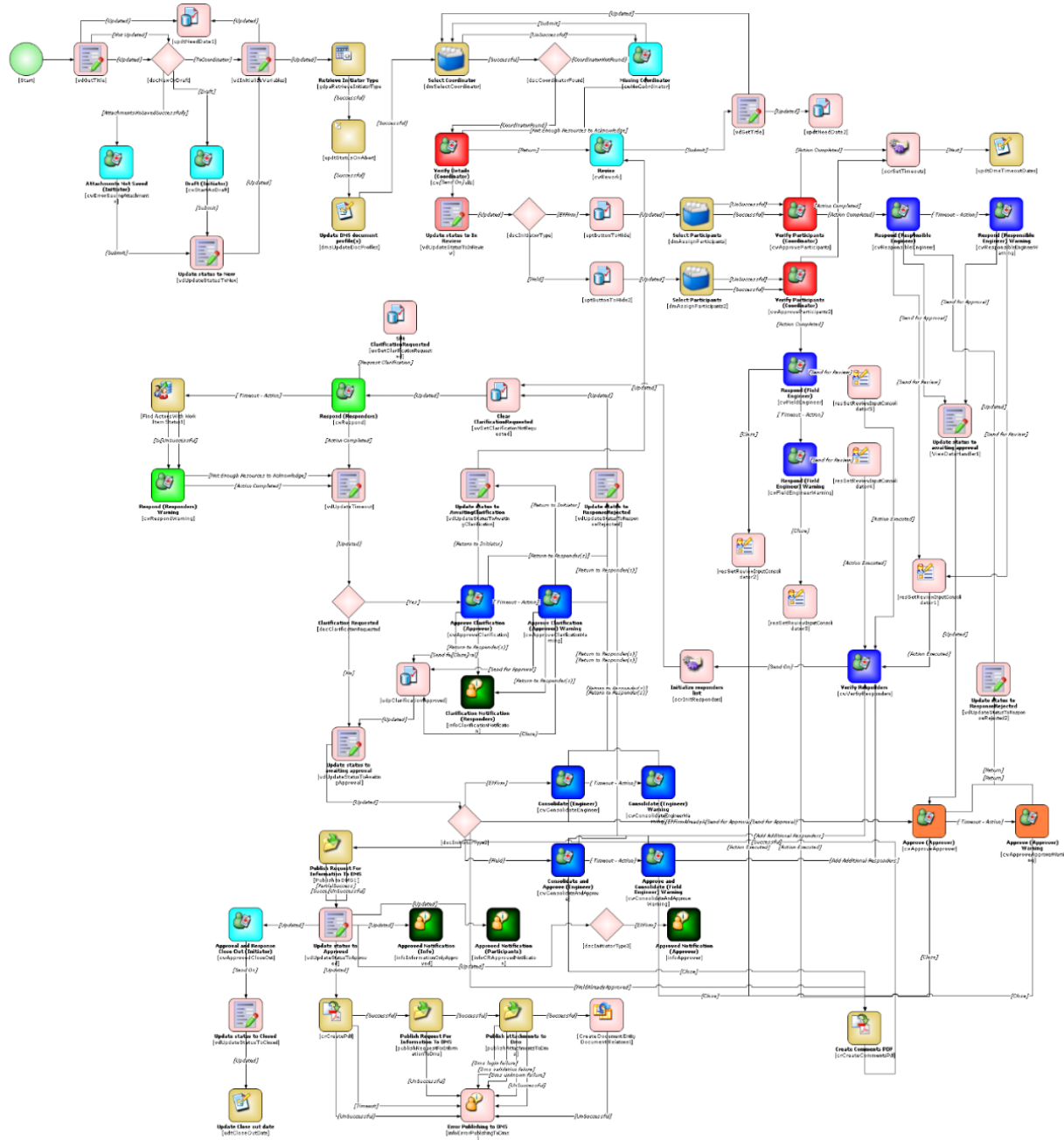


Figure 5-5: An RFI Workflow Process Used in a Capital Mega Project

Enactment of each step of every workflow instance is performed by the workflow engine of the workflow management system, and the execution details and the workflow instance associate with each step are saved in databases. Conducting a discrete event simulation of the RFI process requires real data, such as: (1) inter-arrival time between initiated instances of the RFI process, (2) temporal details linked to the flow of information in each step of the workflow instances, and

(3) the processing time in each activity for every workflow instance. SQL queries were used to access and extract the required data from the databases. The most meaningful extracted fields that are used for the simulation are shown in Table 5-2. A sample of the records containing those fields, imported into a spreadsheet file, is presented in Table 5-3.

Table 5-2: Data Fields and Their Description

DATABASE FIELD	DESCRIPTION OF THE FIELD
WF_ID	Workflow ID
ActivityDisplayName	Task Name in the Workflow
CreatedDateTime	Date and Time Task Created
CompletedDateTime	Date and Time Task Completed
OwnershipDateTime	Date and Time Task Accessed by the Responsible Person
ResponseBy	Date and Time of the Response
CurrentStatus	Current Status of the Request
Name	Responder Name
Version	Workflow Version

Table 5-3: A Sample of Retrieved RFI Workflow Process Enactment Data

WF_ID	ACTIVITYDISPLAYNAME	CREATED DATETIME	OWNERSHIP DATETIME	COMPLETED DATETIME	RESPO NSEBY	CURREN TSTATUS	VER SION
55	Verify Details	3-10-11 1:04:24	3-10-11 14:06:43	3-10-11 14:07:25	NULL	Send On	2
55	Verify Participants	3-10-11 14:07:28	3-10-11 14:07:32	3-10-11 14:08:27	NULL	Send On	2
75	Approve (Approver)	3-16-11 17:03:02	3-16-11 18:58:25	3-16-11 21:31:21	3-28-11 23:00:00	Close	2
75	Approved Close Out	3-16-11 21:31:26	3-17-11 19:17:02	3-17-11 19:18:22	NULL	Send On	2
75	Approved Notification	3-16-11 21:31:27	NULL	NULL	NULL	Deleted	2
75	Approved Notification	3-16-11 21:31:27	NULL	NULL	NULL	Deleted	2
75	Approved Notification	3-16-11 21:31:32	NULL	NULL	NULL	Deleted	2
75	Respond (Engineer)	3-11-11 17:49:00	3-14-11 15:10:03	3-16-11 17:02:57	3-16-11 23:00:00	Send for Approval	2
75	Verify Details	3-10-11 15:17:18	3-11-11 17:45:08	NULL	NULL	Closed	2
75	Verify Details	3-10-11 22:17:18	3-11-11 17:45:08	3-11-11 17:45:34	NULL	Send On	2

WF_ID	ACTIVITYDISPLAYNAME	CREATED DATETIME	OWNERSHIP DATETIME	COMPLETED DATETIME	RESPONSEBY	CURRENTSTATUS	VERSION
75	Verify Participants	3-11-11 17:45:38	3-11-11 17:46:18	3-11-11 17:49:00	NULL	Send On	2
77	Approve (Approver)	3-14-11 19:13:55	3-14-11 19:14:58	3-14-11 19:14:58	NULL	Close	2

The details regarding every step of the enactment of every workflow instance have been stored in the databases, as represented in Table 5-3. For example, row one displays a “Verify Details” activity from a workflow instance of version 2 of the RFI process with workflow ID 55, and includes the temporal details such as created date and time, ownership, and completed date and time. The name of the actors has been omitted from the table for privacy, and the responsibility field, which is typically according to the RACI chart, has not been accurately recorded. The extracted data have been used in simulation of the RFI process using SIMUL8 software.

5.3.2 Modeling and Simulation using SIMUL8

Several simulation software packages are available for conducting discrete event simulations, such as ExtendSim, AnyLogic, FlexSim, Process Simulator, Simio, GoldSim, Promodel, and SIMUL8. SIMUL8 is one of the most popular simulation software packages with several features for modeling and simulation of various types of processes or systems, including business processes. In this section SIMUL8 is used to model and simulate the RFI workflow process depicted in Figure 5-5.

As Table 5-1 presents, this RFI workflow process is comprised of nine versions, each slightly different. Version 8 has been used more than the other versions, and offers the greatest number of workflow instances. Among the total number of 22 840 workflow instances recorded in the database, more than half of the instances (13 233) belongs to this version. Therefore, version 8 was selected for the modeling and simulation.

The core structure of this workflow follows the general steps expressed in the common core structure of RFI workflows in Figure 3-4 of Chapter 3. An RFI is initiated either by a draft request within the RFI workflow or by a request submitted by an external workflow. A coordinator is then assigned to the RFI to verify the details of the request, and to reject it if more details are necessary, or to accept it if it is satisfactory. The coordinator then assigns one or more responders to the

workflow process to respond to the request. If the response takes more than a predefined duration of time, the participant receives a response warning from the system. The responded RFI then should be approved by a consolidator, and is sent out and closed after approval. The approval process is also time restricted and generates an approval warning if takes more than a specified amount of time.

There is, however, one important distinction in regard to the request type; after the *Verify Details* activity, according to the type of the request, the flow is directed to either the construction site (Field) or the head office (Firm). The *Field Request* should be sent to the same company's construction site for more information or the construction site of another company; the *Firm Request* has the same two options. The RFI requests to the external companies exit from the workflow while counted, and the internal requests follow the next steps to be responded, approved, and closed.

For simulation of version 8 of the specified RFI process a number of simplifications and assumptions have been made:

- The workflow management system is always available and the automated activities such as warnings, notifications, and status updates, are enacted by the workflow engine almost instantaneously; therefore, only human-performed activities are modeled.
- All the RFI workflow instances have similar priorities, and therefore the work packages in queues before each activity are executed in chronological order.
- RFI requests are initiated via two different initiation methods: some of the requests are initiated within the workflow using the draft initiation task, several other requests are initiated by external parties and are imported into the RFI workflow. Therefore, the first activity in the workflow to receive all the requests is the *Verify Details* activity. Consequently, for calculation of the arrival rate of the requests to the system, the arrival rate of the requests to the *Verify Details* activity was used. In other words, the distribution of the RFI requests arrival rate to the system is calculated according to the *created date and time* for all the instances of the RFI workflow in the *Verify Details* activity.

Processing time distributions for each activity of the workflow and their parameters were calculated based on the data records of *CreatedDateTime*, *CompletedDateTime*, and *OwnershipDateTime*. The arrival rate distribution, as discussed, was calculated based on the arrival rate for the *Verify Details* activity. The simulation conducted for a trial of 250 runs in which the duration of each run is defined as one year. A snapshot of the simulation model in SIMUL8 is shown in Figure 5-6.

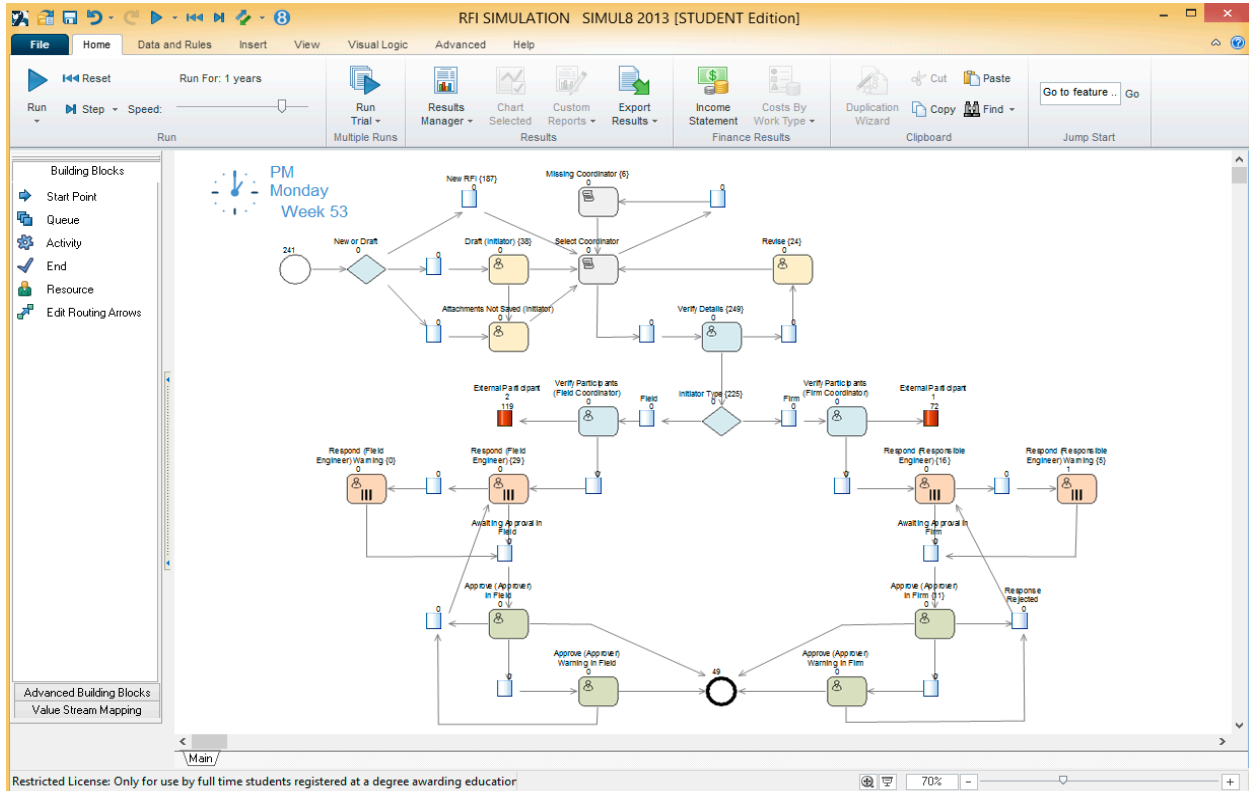


Figure 5-6: A Snapshot of the Simulation Model in SIMUL8

5.3.3 Simulation Analysis

Simulation of the RFI process with real data from a capital project provided insights about the behavior of the RFI process. Some of the metrics that were used to analyze the behavior of the process are: the number of RFI processes issued during a specific period of time, the number of people that are assigned to each role to perform manual tasks of the workflow, the average amount of time it takes for each role in the workflow to perform its tasks, and the average amount of time for the whole system to complete an RFI process.

Detailed analysis of the model revealed particular bottlenecks that led to a few process improvement suggestions. For example, the *Verify Details* activity is a critical bottleneck in the flow of the documents within the simulated model. At particular points in time, the documents are accumulated in a queue before this node, and it takes a substantial amount of time for those documents to be processed. Adding another actor to this role will decrease the processing time of this activity and will make the whole system more efficient and ready to handle the workload. In addition, based on the analysis, the route with the *Attachments Not Saved* task is redundant. Handling missed attachments is performed within the *Verify Details* activity, which is why the *Attachments Not Saved* automated task has never been used. Therefore, it can safely be eliminated from the workflow.

Discrete event simulation (DES) is a valuable tool for analyzing the behavior of workflow processes and providing suggestions for improvements. DES initially considered as a validation tool to analyze the behavior of existing and the IFP processes. However, comparing the performance and efficiency of existing and IFP processes requires real data from both processes. Process performance measurements for comparing workflow processes focus on quantitative key performance indicators, such as workflow capacity, average handling time, average wait time, and rate of completion, that are not measurable without real data. Therefore, the simulation was not used for IFP processes or as a validation tool for this research.

5.4 Discussion

As outlined at the beginning of this chapter, four approaches were selected for validating the results of this research initiative. With the progress of the research, and taking into account the flexibilities and limitations of each method, expert feedback and functional demonstration turned into the main validation approaches of the research. Process analysts, computer science experts, IT specialists, and construction management professionals have provided valuable feedback on each stage of the research. In addition, functional demonstration of the deployment method of the IFP systems, and the value and benefits of its application – conformance and interoperability – has been the other main validation approach.

Discrete event simulation (DES) was the first validation method that was investigated and examined. An RFI workflow process, which was used in a recent mega project in Canada, modeled

in the SIMUL8 simulation software, and real data from a capital project were used to examine the model and analyze the results. This effort, however, proved that the discrete event simulation is not a suitable validation approach for this research with two justifications. (1) In the DES model, the Key Performance Indicators (KPI) for measuring the performance of the model are defined for human-performed tasks, but not for the automated activities of the workflow process. The automated activities are enacted almost instantaneously, and do not have a significant impact on the DES model. Since the DES should have been used for comparing the efficiency of an existing process and an IFP, in which the structure of the processes and the automated tasks are the main difference, the DES modeling was determined to not be an appropriate validation methodology. (2) The real data for an existing process were available, but not for an IFP process. Therefore, a valid comparison between the real process performance and the IFP process conformance could not be performed using a DES approach.

Surveying of industry experts was considered as a validation methodology, but was ineffective and was not pursued. The primary reason is the interdisciplinary nature of the research. Different types of surveys should have been prepared in particular stages to address different aspects of the research. For example, a wide range of potential responders: process analysts, computer science and IT experts, and construction management professionals, should have been involved in the surveys, each assessing partial benefits of the IFP system. This approach would have been non-comprehensive, unreliable, and impractical, and was disregarded.

Chapter 6

Improving Process Conformance with IFP

6.1 Process Conformance

Conformance is defined as compliance with practices, standards, rules, or established behavior. Conformance has different aspects and can be specified with various considerations and on different levels. Examples of levels include conformance with industry best practices, industry regulatory, corporate, business unit, project, or contract. With the widespread use of process-aware information systems, conformance of processes to industry best practices, corporate rules and regulations, or service level agreements is becoming increasingly important.

In the literature, process conformance describes identification and examination of the differences or discrepancies between a process model and the behavior of the executed version of the process (Mannhardt, Leoni, Reijers, & Aalst, 2015). Process compliance refers to the relationship between the specifications for executing a business process and the specifications regulating a business (Governatori & Sadiq, 2009). Conformance or compliance checking techniques formally present that business processes comply with relevant constraints such as regulations, laws, or guidelines (Ly, Maggi, Montali, Rinderle-Ma, & van der Aalst, 2015). In this research, workflow conformance checking is used more specifically to examine the conformity of customized workflow processes with the IFP processes that are a representation of the accepted specifications based on best practices.

With the widespread use of information systems, automated workflow processes are used for implementation of practices, regulations, and contractual obligations, and thus to facilitate conformance. Processes are customized in each organization based on their limitations and requirements. For example, processes in a construction project are customized and implemented in an EPPM system according to the organizational structure, and the unique project characteristics, such as: size, delivery method, and execution plan. Therefore, practice implementation through EPPM systems improves conformance through automation and transparency, but the required customization generally works against best practice conformance.

A potential solution is introduced based on the industry foundation processes (IFP) construct presented in this research.

6.2 Workflow Conformance Checking

Conformance checking techniques typically focus on the control-flow of a process, analyzing the order of the steps involved, to determine the conformance of the process with the expected behavior (Mannhardt et al., 2015). There are two primary types of conformance checking: (1) forward conformance, in which the restrictions are enforced in the process design stage to prevent designing a non-conformant process; and (2) backward conformance, in which the steps and flow of work in an implemented process are examined to discover non-conformant behavior (Taghiabadi, Fahland, Dongen, & Aalst, 2013).

This chapter establishes a foundation for workflow conformance checking by developing an algorithm for analyzing the control-flow of workflow processes using a first-order logic language – Alloy. In addition, a Java-based tool was developed to automate the conformance checking process. The algorithm was applied to different examples of the conformant and non-conformant workflow processes, and then, a real-world RFI process was modeled as a case study to validate the accuracy of the algorithm and to demonstrate the functionality of the workflow conformance checking tool. The case study is an example of the backward conformance checking approach; to identify non-conformant behavior of an implemented workflow process. The same workflow conformance checking mechanism can be used in a workflow design tool, as a forward conformance checking methodology, to promptly notify the designer of any non-conformant design of the workflow process.

6.3 Conformance Checking Algorithm

Conformance of a customized workflow process to a specification workflow process, such as an IFP can be performed by analyzing and comparing the structure and the control-flow of the customized workflow against the specification workflow, and considering conformance criteria such as: the required presence of core activities, the required sequence of core activities, the required level of authorization, the required sequence of authorization (hierarchy), and the required completion time of activities.

In this section, graph theory concepts are employed to formally define workflow processes in terms of directed graphs, with nodes and edges of a directed graph representing the activities and their relationships respectively, in a workflow process. The workflow inheritance rules outlined in Chapter 4 are expressed in terms of computer science and graph theory concepts such as: graph dominators, dominator tree, immediate dominator, and post-dominator. The dominators concept applies to directed graphs with distinguished start and end nodes (Georgiadis, Tarjan, & Werneck, 2006; Lengauer & Tarjan, 1979; Prosser, 1959). A well-formed graph must be connected – every node must be reachable from the start, and the end must be reachable from every node. In addition, the start must not be reachable by any node, and the end cannot be reached by any node. These conditions are true of well-formed computer control flow graphs and also of workflow processes represented by directed graphs. Figure 6-1 shows examples of directed graphs that are not well-formed (Tao Lue Wu, 2015).

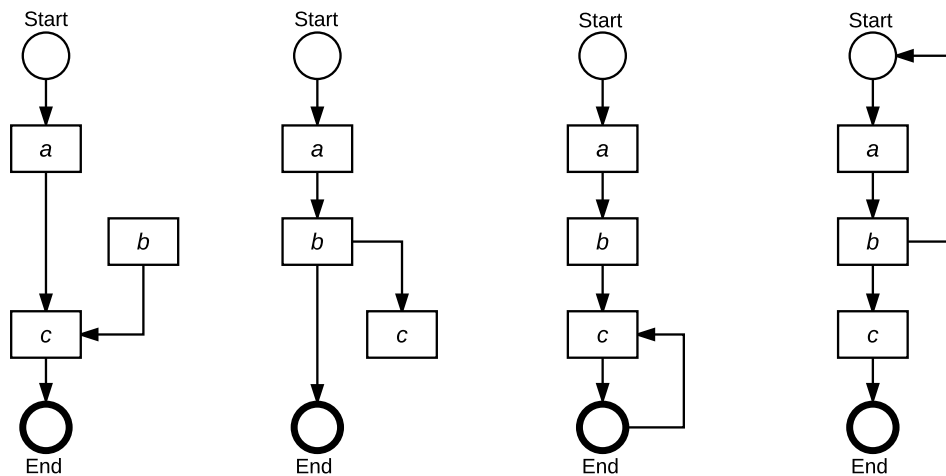


Figure 6-1: Examples of Directed Graphs That Are Not Well-Formed

The left-most graph in Figure 6-1 is not well formed because node *b* is not reachable from the start, and the center-left graph is not well formed because the end is not reachable from node *c*. The center-right graph is not well formed because the end node reaches to node *c*, and the right-most graph is not well formed because the start is reachable from node *b*.

In a directed graph, a node *d* dominates a node *n* if every path from the start node to node *n* must go through node *d*. Similarly, a node *p* post-dominates node *n* if every path from *n* to the end node must go through *p* (Georgiadis et al., 2006; Lengauer & Tarjan, 1979; Prosser, 1959). The *immediate dominator* is the dominator closest to the node. Similarly, the *immediate post-*

dominator is the post-dominator closest to the node. Graph dominators can be computed in quadratic time (Lengauer & Tarjan, 1979).

The dominance concept includes particular properties that are used for developing an algorithm for comparing the structure of a customized workflow with a specification workflow based on the workflow inheritance rules. For example, the dominance relation is reflexive, anti-symmetric, and transitive. This means that: (1) any node dominates itself, (2) if a dominates b and b dominates a then a is equal to b , and (3) if a dominates b and b dominates c then a dominates c . For more information about the theoretical principles of the developed conformance checking algorithm please refer to Tao Lue Wu's report which is partially presented in Appendix C (Tao Lue Wu, 2015).

A customized workflow is said to conform to a specification workflow if the following three conditions are met (Golzarpoor, Haas, & Rayside, 2016):

1. The customized workflow contains all of the steps (nodes) in the specification workflow.
2. For every step X that exists in both the specification and customized workflows, X 's immediate dominator in the specification workflow is one of its dominators in the customized workflow.
3. For every step Y that exists in both the specification and customized workflows, Y 's immediate post-dominator in the specification workflow is one of its post-dominators in the customized workflow.

These conditions formalize the intuitions that steps in the specification workflow cannot be skipped, and that new steps may be added. Using these conditions, an edge e from source node s to target node t is classified as a *skip* edge if either the source node s fails to meet condition (2) above, or the target node t fails to meet condition (3).

Accordingly, an algorithm to assess the conformance of a customized workflow with a specification workflow is developed as follows (Golzarpoor et al., 2016):

1. Confirm that both the specification workflow and the customized workflow are well-formed. If not, report malformed workflow and terminate.

2. Confirm that the customized workflow contains all of the steps in the specification workflow (condition 1). If not, report non-conformance due to step deletion and terminate.
3. Compute dominators and post-dominators for every node, in both the specification workflow and the customized workflow.
4. For every node that exists in both specification and customized workflows, confirm that the immediate dominator in the specification workflow is still a dominator in the customized workflow (condition 2 above). If not, report edges that terminate at such nodes as skip edges.
5. For every node that exists in both specification and customized workflows, confirm that the immediate post-dominator in the specification workflow is still a post-dominator in the customized workflow (condition 3 above). If not, report edges that originate at such nodes as skip edges.
6. If no skip edges, then report conformance.
7. Terminate.

This algorithm has been implemented in the Alloy declarative specification language (Daniel Jackson, 2011), so that specific pairs of workflows can be automatically checked for conformance using the associated Alloy Analyzer tool. An alternative implementation could be written in a conventional imperative programming language (*e.g.*, Java, C, *etc.*) using one of the well-known algorithms for graph dominators (*e.g.*, Georgiadis et al., 2006; Lengauer & Tarjan, 1979).

6.4 The Alloy Language and Its Advantages

Alloy is a first-order logic programming language with sets, relations, and transitive closure. It is typically used for writing specifications of rich, graph-like data structures, which are structurally similar to workflows. The Alloy Analyzer translates the Alloy first-order logic to propositional logic (*i.e.*, Boolean formulas) by providing finite bounds for the quantifiers. If the finite bounds used for translation are insufficient, then the resulting Boolean formula is an approximation of the original first-order formula. For example, if the original formula quantifies over an infinite set, such as the integers, then the bounds will be insufficient. Since workflows are always finite

structures, and from a computational standpoint not particularly large, the bounds for the translation will always be sufficient for workflows.

Alloy has three advantages over a conventional imperative language for workflow conformance checking. First, the Alloy language is designed for working with rich graph-like structures, whereas conventional imperative programming languages are not (Schwartz, Dewar, Schonberg, & Dubinsky, 1986). Second, the Alloy Analyzer includes a visualizer for inspecting the inputs, outputs, and state of the program. Third, in addition to running the program with specific inputs, the Alloy Analyzer can also automatically generate test inputs for sub-procedures or the program as a whole.

The computational complexity of computing dominator trees is quadratic (Lengauer & Tarjan, 1979), which is within the expressiveness of Boolean formulas (NP-complete (Cook, 1971)). Therefore, the Boolean formula produced by the Alloy Analyzer is an accurate representation of the problem of computing the conformance of two workflows. Modern Boolean Satisfiability solvers routinely solve formulas with tens of thousands of variables and hundreds of thousands of clauses. The Boolean formulas produced for workflow conformance checking typically have several thousand variables and several thousand clauses, and solve in a few tenths of a second using MiniSAT (Eén & Sörensson, 2004) on an old laptop (AMD A4-3300M processor running at 1.9GHz; manufactured in 2011). Workflow conformance checking is well within the capabilities of modern SAT solvers.

In software engineering, Alloy is used for analyzing software designs, including analyzing imperative programs for conformance with their logical specifications (Dennis, 2009). The workflow-specification conformance problem is similar to – but importantly different from – the program-specification problem: most importantly, workflow conformance checking is only concerned with the arrangement of the steps, and not with the outputs of the workflow. Program-specification checking is concerned with the outputs computed by the program. Workflows involve highly trained people exercising professional judgments in complex real-world situations, rather than computers merely following instructions. The workflow-specification conformance problem is similar to the subgraph isomorphism problem (Ullmann, 1976); are the steps of the specification workflow embedded in the customized workflow in a way that preserves

their ordering? The subgraph isomorphism problem is simplified here by fixing the node correspondences based on the node labels. Order preservation is relaxed from the subgraph isomorphism problem by permitting the insertion of nodes and the insertion and removal of edges. Permissible order-preserving modifications are formalized in terms of dominators and post-dominators.

6.5 Workflow Conformance Checking using Alloy

Alloy is a declarative programming language. In a declarative programming language, a model is built upon a description of the behavior of the system, without defining the mechanisms for that behavior. The more constrained the description of the system, the more limited are the behaviors. This allows very concise models to be constructed and analyzed (D. Jackson, 2002).

```

open util/graph[Step]
abstract sig Step {
  -- edges in the Contractor's workflow
  v : set Step,
  -- dominator tree
  idom2 : one Step,
  ipostdom2 : one Step
}
abstract sig Foundation extends Step {
  -- edges in the IFP workflow
  w : set Foundation,
  -- dominator tree
  idom1 : one Step,
  ipostdom1 : one Step
}
-- distinguished Start and End nodes
one sig Start, End extends Foundation {}

```

Figure 6-2: Alloy Implementation of Workflow Process

Figure 6-2 shows the Alloy implementation of workflow process, and Figure 6-3 presents the Alloy implementation of a well-formed workflow process. *Step* is used to denote the set of all nodes that are involved in the model and *v* is used to denote the set of all edges that are incident with nodes in *Step*. Two edges are called incident, if they share a node. *Foundation* and *w* are used to model a workflow. *Start* and *End* are predefined as nodes in *Foundation*. *Foundation* is a subset of *Step* and *w* is the set of all edges that are incident with nodes in *Foundation*. *w* and *v* are independent. In the Alloy implementation, *nodes* denote the set of all nodes (i.e. a set of *Step* in Alloy) and *e*

denotes the set of all edges in the workflow. In this case, the edges are represented by a binary relation between two nodes (i.e. $e : Step \rightarrow Step$) (Tao Lue Wu, 2015).

```

pred wellFormed[nodes : set Step, e : Step->Step] {
  -- nodes includes Start and End
  Start in nodes
  End in nodes
  -- all nodes are reachable from Start
  nodes in Start.*e
  -- Start has no incoming edges
  no e.Start
  -- End is reachable from all nodes
  nodes in *e.End
  -- End has no outgoing edges
  no End.e
}

```

Figure 6-3: Alloy Implementation of Well-Formed Workflow Process

Figure 6-4 shows an excerpt of the Alloy code for workflow conformance checking that performs the identification of skip edges based on the dominator analyses. An expression such as $n.idom1$ evaluates to the immediate dominator of node n in the specification workflow. An expression such as $n.^idom2$ evaluates to the set of all dominators of n in the customized workflow. Here $idom1$ and $idom2$ are functional binary relations that map nodes to their immediate dominator in the specification or customized workflow, respectively. The caret (^) operator computes the transitive closure of a binary relation (i.e., finds the entire set of dominators). This code is more succinct in Alloy than it would be in a conventional imperative programming language (Golzarpoor et al., 2016).

```

fun skips[] : Step -> Step { { s,t : Step |
  -- source -> target is an edge in the customized workflow and
  s->t in v and (
    -- target's original immediate dominator is not in its new dominators
    t.idom1 not in (s + t.^idom2)
    or
    -- or source's original immediate post-dominator is not in its new post-dominators
    s.ipostdom1 not in (t + s.^ipostdom2)
  ) } }

```

Figure 6-4: Alloy Specification (Excerpt) of Workflow Conformance for Steps 4 and 5 of the Algorithm
 The Alloy conformance checking algorithm has been applied to several different test cases. For example, Figure 6-5 shows the Alloy visualization of the conformance check of example workflow W9 from Figure 4-3. Figure 6-5(a) shows the specification workflow ($A \rightarrow B \rightarrow C \rightarrow D$).

Figure 6-5(b) shows the customized workflow (W9 in Figure 4-3). Figure 6-5(c) shows the conformance analysis. The gray nodes are those that exist in both the specification and customized workflows. The white nodes are new nodes in the customized workflow. Black edges are those that exist in both workflows. Grey edges exist in the specification workflow, but have been deleted in the customized workflow ($B \rightarrow C$). Green edges are new legal forwards edges in the customized workflow ($B \rightarrow N$, $N \rightarrow P$, and $P \rightarrow C$).

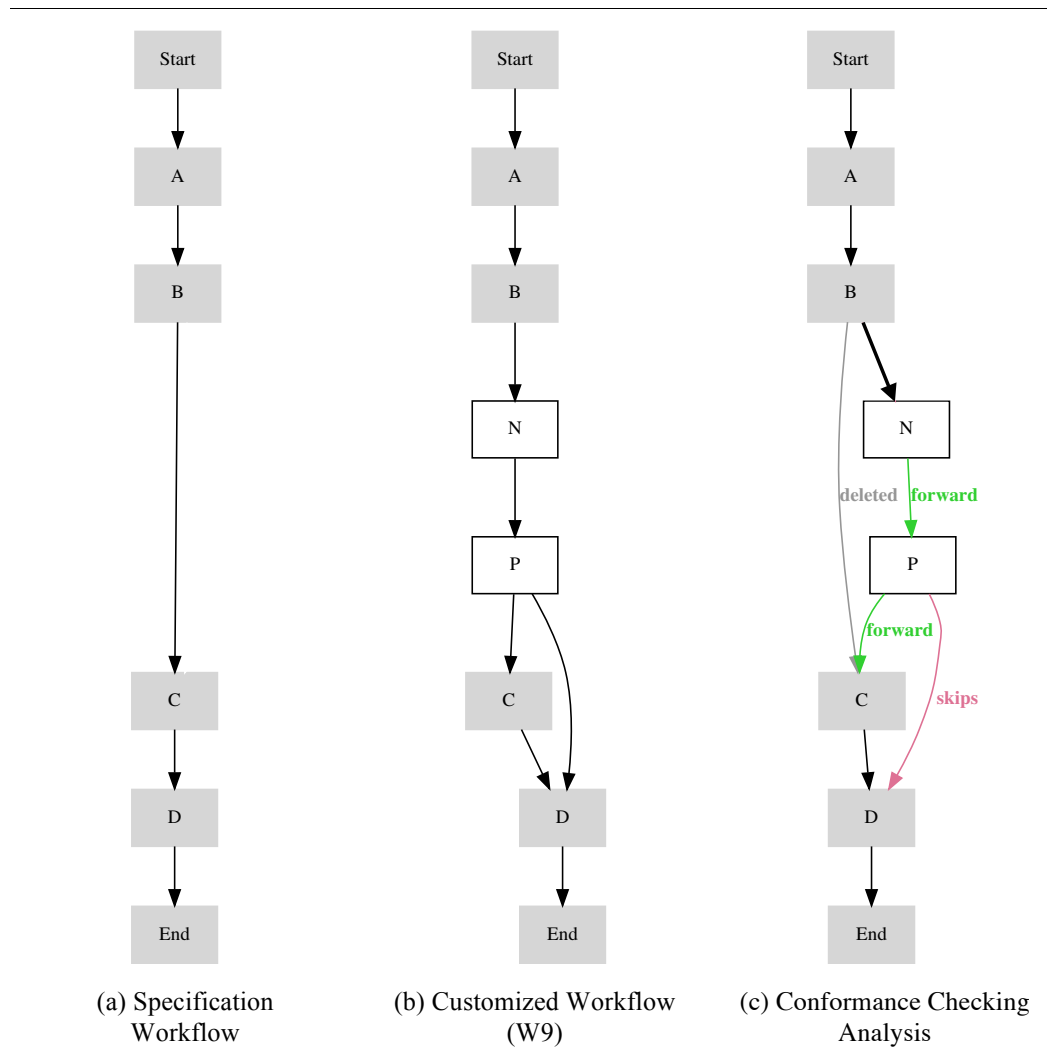


Figure 6-5: Visualization of Conformance Checking for Workflow W9 of Figure 4-3

As discussed in Chapter 4, workflow W9 does not conform to the specification workflow. This is illustrated in the analysis by the red skip edge ($P \rightarrow D$). The dominator subset analysis reports that edge $P \rightarrow D$ has a problem. Without edge $P \rightarrow D$ the D 's dominators in the specification workflow

are A , B and C , and in the customized workflow are A , B , N , P , and C ; whereas with the $P \rightarrow D$ edge D 's dominators in the customized workflow are A , B , N , and P : it is acceptable to add N and P , but not to remove C . The post-dominator subset analysis reports that edge $P \rightarrow D$ has a problem. Without $P \rightarrow D$ edge the B 's post-dominators in the specification workflow are C and D , and in the customized workflow are N , P , C , and D ; whereas with the $P \rightarrow D$ edge, B 's post-dominators in the customized workflow are N , P , and D : it is acceptable to add N and P , but not to remove C . As a consequence of the dominator and post-dominator analysis, the edge $P \rightarrow D$ is reported as a skip edge.

6.6 Validation Case Study

The developed workflow conformance checking algorithm is demonstrated in this section to check the conformance of the more detailed, customized RFI workflow processes presented in Figure 4-4. The code listing for the RFI workflow conformance checking is presented in Appendix D. The RFI workflow process shown in Figure 4-4(a) is in conformance with the IFP process presented in Figure 4-2, and the Alloy implementation correctly identifies it as a conformant workflow. The customized RFI workflow process in Figure 4-4(b) is a non-conformant workflow process. The result of the conformance checking analysis for this workflow is shown in Figure 6-6.

The visual conventions in Figure 6-6 are similar to the conventions in Figure 6-5: gray nodes are those in the specification workflow; white nodes are those added in the customization; black edges exist in both workflows; gray edges are those that have been removed in the customization; green edges are new forward edges; blue edges are new back edges; red edges are skips.

One of the purposes of this customization was to enable direct response of the Coordinator to the request by adding the path from the “Respond Directly” activity to the “Response Close Out” activity, bypassing the responders and the consolidator. Since bypassing steps is not permitted by the inheritance rules, this customization is identified as non-conformant with the specification (Figure 4-2). The Alloy implementation correctly identifies the edge from “Respond Directly” to “Response Close Out” as the skip edge.

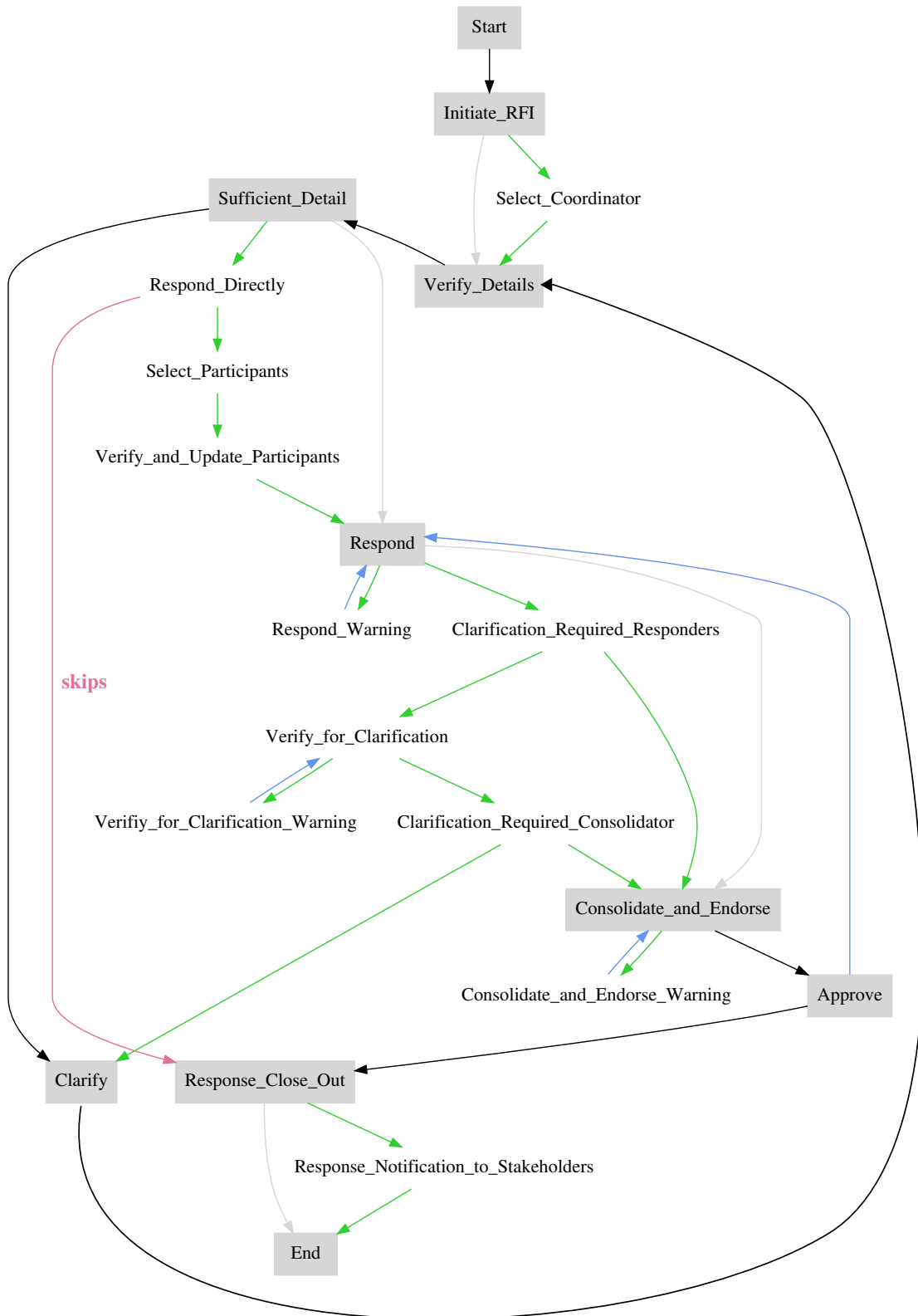


Figure 6-6: Conformance Checking Analysis of a Non-Conformance RFI Workflow Process

6.7 Automated Workflow Conformance Checking Tool

The conformance checking algorithm developed with the Alloy language enables the analysis and the conformance checking of any customized workflow process with a specification workflow; however, the customized and the specification workflows should be modeled in Alloy syntax. It would be very time-consuming and error-prone to manually convert customized and specification workflows to Alloy accepted format. Therefore, two Java applications were developed to streamline the conformance checking process.

Windows Workflow Foundation (WF), as a component of the Visual Studio, includes a Workflow Designer. The Workflow Designer is used for developing WF workflow processes, which are stored as XAML files – a declarative markup language. A Java application was developed to parse the contents of the XAML files and translate them to the proper format accepted by the Alloy Analyzer. This application automatically identifies whether the workflow is in FlowChart format or StateMachine format (0, 5.2.2), and translates them into an Alloy file (.als). The documentation for the Translator application is presented in Appendix E, and the code listing is revealed in Appendix F.

In addition, an Automator application was developed which takes the .als files and visualization theme files (.thm) of the specification and customized workflow processes, send them to the Alloy Analyzer for conformance checking, and provide the final analysis result in a visualization file (.dot). The visualization result can be viewed by the Graphviz application. Figure 6-6 is a sample of the visualization that was generated automatically. The documentation for the Automator application is presented in Appendix G and the code listing is displayed in Appendix H.

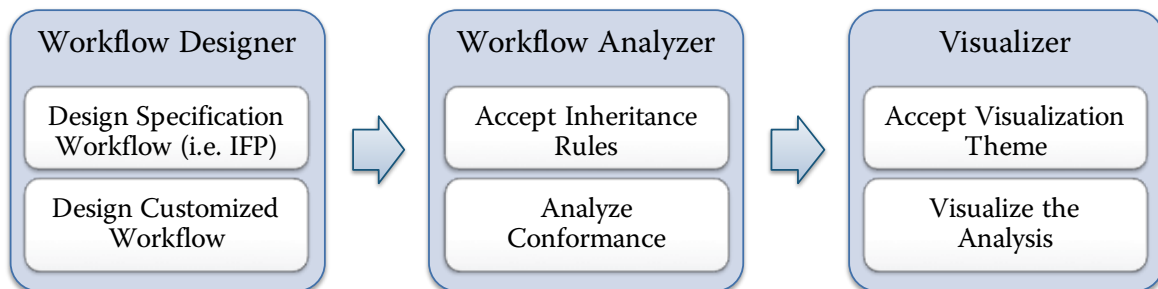


Figure 11: Three Components of the Developed Conformance Checking Tool

In summary, the automated workflow conformance checking tool is comprised of three components: (1) Workflow Designer, (2) Workflow Analyzer, and (3) Visualizer, which work together to streamline the process of workflow conformance checking. Specification and customized workflows are designed in Visual Studio Workflow Designer and are stored as XAML files. The Translator.java application converts XAML files (state-machine or flowchart) to Alloy format. The Analyzer uses the developed Alloy algorithm to analyze and compare workflows using inheritance rules and determine conformance or non-conformance of the customized workflows compared to the specification workflow. The result of the analysis is then displayed via the Visualizer.

Chapter 7

Improving Process Interoperability with IFP

7.1 Process Interoperability

Process interoperability is the interaction and collaboration of workflow processes between different organizations. Process interoperability facilitates cross-organizational cooperation and exchange of information to achieve a common goal. Usually, it is required in client-supplier relationships, or in partnership situations, where workflow processes from different organizations connect and exchange information.

Process interoperability is closely related – but is distinct from – process integration. The focus of process integration is intra-organizational, and the focus of process interoperability is inter-organizational process interactions. Process integration or orchestration is the management of workflow processes within one organization or business unit, and is typically controlled by a single workflow engine. Process interoperability or choreography is the collaboration and management of interactions among workflow processes from different organizations or business units, and is the interaction of processes that are controlled by separate workflow engines.

Process interoperability is not limited to the connection of workflow processes and the flow of information between different organizations. It requires communicating the purpose and the structure of each workflow process to the other, and understanding how each collaborating company operates. It is a consistent approach to defining and managing arrangements between processes that expand over multiple organizations. Process interoperability typically focuses on the common processes. Many components of the common processes are similar, and there are equivalent components in other processes. Understanding this similarity enables the reuse of process components, and facilitates improved collaboration between processes (“The Australian government business process interoperability framework,” 2007).

Process interoperability facilitates interaction of workflow processes by providing: (1) alignment between workflow processes via sharing the process structure, and updating the flow of activities and the role of participants in each organization; (2) efficiency through the reuse of proven

practices that are implemented as processes; (3) security and privacy by sharing only a high-level public view of the processes, abstracting from the proprietary details; and (4) stability by sharing the current status of the interacting workflow processes, so that any interruption in one process is communicated to the other. These are what industry foundation processes offer, and are demonstrated in this chapter for some of the common workflow processes in the construction industry.

7.2 Process Interoperability Approaches

There are three different approaches for interoperation of workflow processes (Chen, Doumeingts, & Vernadat, 2008; Chen, Vallespir, et al., 2008):

1. Federated approach, in which there is no common structure or standard format between the components of the interoperating processes. In this approach, companies must start to identify all the components that are required for interoperation, and accommodate interoperation based on an agreement. This approach is costly, difficult, and time-consuming.
2. Unified approach, in which a common model at a meta-level is available. The meta-level model is not a structured model, but offers a method of mapping between the components of processes, manually or by means of semantic equivalence.
3. Integrated approach, in which an accepted model with a common structure and specific predefined components is available. This predefined structured model is not a standard but is accepted by all process stakeholders. This model facilitates essential interoperability between workflow processes, and is the starting point to expand and build on the additional required components.

The IFP system offers a structured model and the required components for improving process interoperability between workflow processes via an integrated approach. The IFP interoperability model proposed in this chapter facilitate exchange of information between workflow processes by defining essential interfaces. Additional identified components that are required for interoperability, and are not part of the IFP, are added to the model through a combination of unified and federated approach.

7.3 Process Interoperability in AEC/FM Domain

The life-cycle of any large-scale construction project consists of distinctive phases from requirements collection, and feasibility study; to preliminary design, detailed design, construction, commissioning, and operation. A large amount of data collection and information exchange occurs between the owner and the project consultant within the Front End Planning (FEP) stage, at the beginning of the project within feasibility, concept, and detailed scope definition sub-phases. This information is extensive in nature and is incorporated into project documents. Additional Information is accumulated during the next phases of the project, and is transferred to several other project stakeholders, such as consultants, general contractors, sub-contractors, vendors, and suppliers, to be handed over at the end of the project to the operation team. Interoperability in the AEC/FM domain facilitates the flow of this extensive amount of information and project documents, among project collaborators and stakeholders, throughout the project life-cycle (Construction Industry Institute, 2015).

Currently, management of large-scale construction projects is almost entirely process-based. Different aspects of a project are governed by sets of workflow processes that are carefully crafted for their specific purposes. Workflow processes employed in management of a construction project are considered a key component of project success. They enable exchange of project documents and information, and facilitate communication and collaboration among project stakeholders. Table 7-1 presents a sample of common workflow processes in a large-scale construction projects. As such, interoperability in AEC/FM domain needs to be facilitated through employment and interaction of workflow processes.

Table 7-1: A Sample of Common Workflow Processes in Large Construction Projects

PROCESS	PURPOSE
Change Management	Integrated project change control for handling of change requests
Request for Information (RFI)	Query for information and controlled response, review, and approval
Transmittal and Submittal	Managing information exchange between project document control and external parties such as clients, vendors, and contractors
Design Review and Approval	Ensures a new set of documents go through a defined review and approval process
Procurement Management	Manages the procurement process document exchange and approvals
Materials Management	Ensures that the materials and equipment are obtained at a reasonable cost, and are available when needed

Contract Management	Pre-award & post-award contract administration
Risk Management	Identifying and managing project risks
Interface Management	Controlling interfaces and managing collaboration between scopes of work
Deliverables Management	Tracking progress of project deliverables

Process interoperability in the domain of the construction industry is the seamless exchange of information between workflow processes, to facilitate exchange of project documents and information between project stakeholders. The IFP interoperability model addresses interoperability from the perspective of internal process standardization and conformance, corporate process and practice assurance, and interface management between stakeholders.

7.4 IFP Interoperability Model

Process interoperability is a vital component of cross-organizational alignment and effective collaboration through workflow processes. However, sharing the details of complex workflow processes and usually proprietary specifications of organizational workflow processes to establish a seamless linkage between workflow processes of organizations is difficult. Organizational processes are part of the intangible assets of each organization. Organizations are not willing to disclose the details and specifications of their organizational processes, but they can share a high-level outline of their workflow processes to establish interaction with other organizations' processes. This is the approach of the IFP interoperability model.

The IFP interoperability model offers a unique solution to facilitate process interoperability between common and high volume construction industry workflow processes, such as design review, change request, request for information, and inbound outbound transmittals. A complex workflow process that is in conformance with an IFP includes all the core activities of the IFP and conforms to the structure of the IFP. The IFP interoperability model facilitates process interoperability by defining the high-level IFP structure of such processes as their public view, and using this public view as the means for all the interactions and communications between processes. An interface is defined for information exchange between processes according to the IFP ontology, to facilitate exchange of different classes of data and information.

Figure 7-1 is an illustration of the IFP interoperability model for two RFI processes, and Figure 7-2 shows it for the interaction of a CR process and an RFI process.

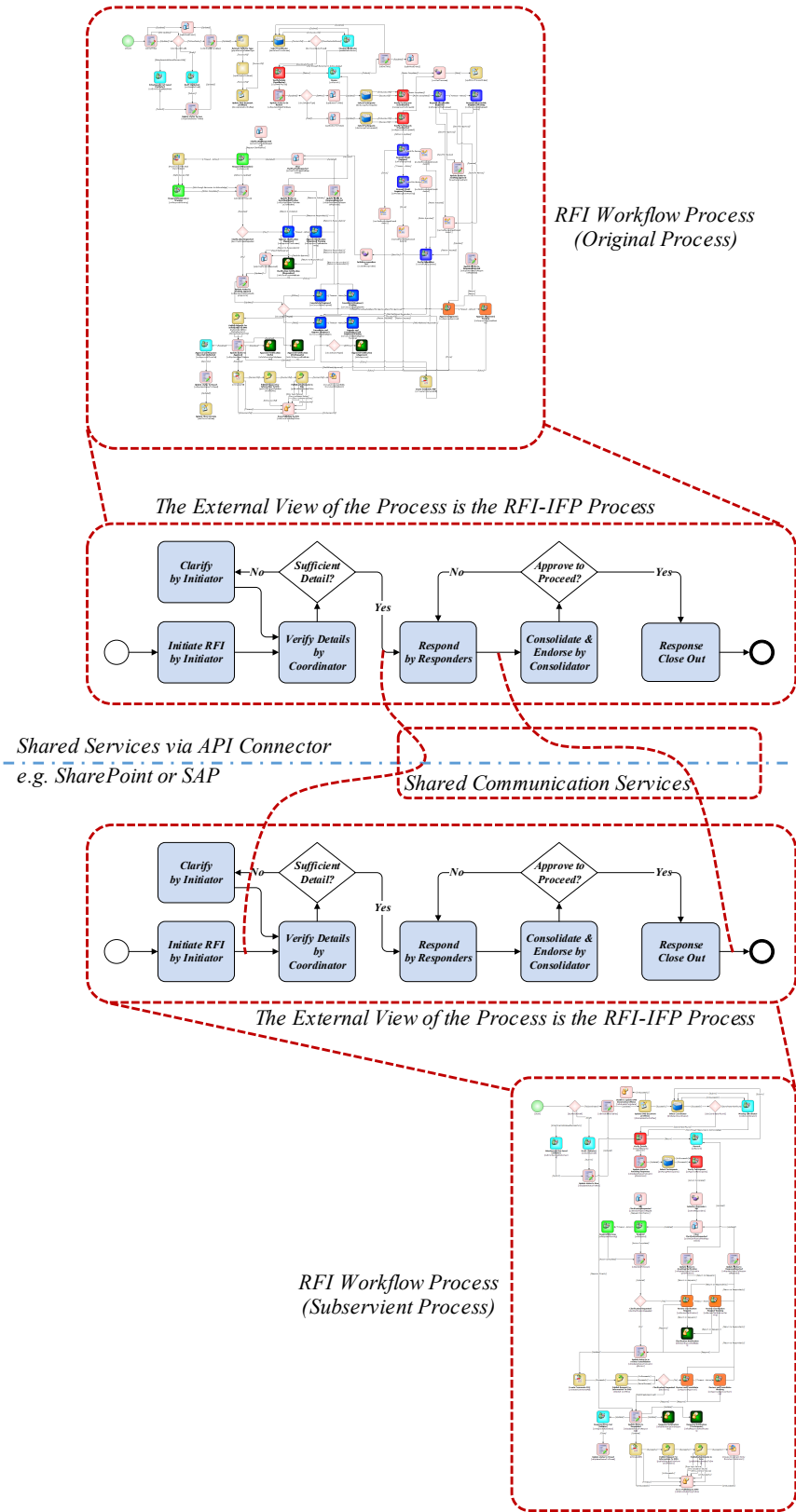


Figure 7-1: IFP Interoperability Model for Interaction of Two RFI Workflow Processes

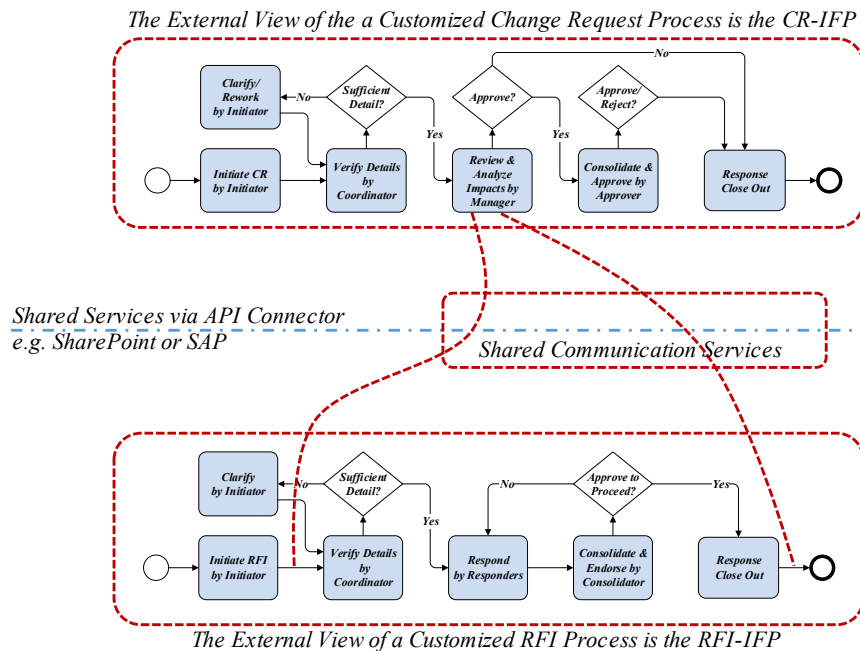


Figure 7-2: IFP Interoperability Model for Interaction of a CR and an RFI

The IFP interoperability model employs the IFP ontology components including the core structure, the abstraction level, the data structures, and the three categories of workflow inheritance – structural, organizational, and temporal – to facilitate interaction of workflow processes by:

- Sharing the high-level structure of the interacting workflow processes i.e. IFP as their public view, which includes core activities and their relationships;
- Sharing the roles and the hierarchy of authorization associated with the high-level structure of the interacting processes;
- Defining the time-bounds and the response due dates for interactions;
- Exchanging data sets that are important for understanding or accurate interpretation of the context of communication, such as general project, process, and contract information, e.g. Project ID, project phase, discipline, and contract ID; and
- Communicating any state change or status update between the high-level structure of the interacting processes.

A workflow process in any step can initiate another workflow process, which is referred to as the subservient process. The initiation is an important step in which the high-level structure of

workflow processes, the hierarchy of authorization, the time-bounds and due dates, and different data sets are exchanged; first from the original process to the subservient process, and then from the subservient process to the original one. The original and subservient processes continue interaction and communication after the initiation until the subservient process is completed and closed out. The subservient process can initiate another process, and this order can continue, but the subservient process is closed out before the original process resumes.

The communication between workflow processes is performed completely through message flows. The flow of messages is separate from the control-flow, which controls the execution order and current state of the process. The control-flow between the steps of any workflow process is controlled by a workflow engine, and the control-flow of one workflow process cannot be transferred to another workflow which is managed by another workflow engine. BPMN 2.0 process modeling standard uses pools to represent all the processes internal to one organization and the sequence control-flow between the activities, and defines the communication between different organizations' processes through message flow between the pools.

7.5 Implementation Using Workflow Foundation (WF) Technology

Microsoft Workflow Foundation (WF) technology, which was used in Chapter 5 to implement the request for information (RFI) workflow process, is used in this section to demonstrate process interoperability between two RFI workflow processes. The implementation is according to the model presented in Figure 7-1.

The RFI workflow process is defined as a state-machine model, each activity is represented as a state. A state class is used to define the RFI core activities, which are inherited by customized RFI activities that add to or modify the functionality of core activities. The RFI-IFP class is defined using the core activities. Customized RFI workflows inherit the functionality of the RFI-IFP class and use customized RFI activities to enhance the functionality of the core processes by implementing additional states or modifying existing ones. Figure 7-3 represents this model which is used for the RFI implementation.

An ideal situation for demonstrating process interoperability is to implement the interacting workflow processes in separate systems that are managed by different workflow engines. In real

circumstances, the technologies of these systems might be different and the technical and information interoperability might need to be dealt with first, before addressing process interoperability. In this case study, however, an Integrated approach is used to reduce the complexity. Two identical customized RFI workflow processes (Figure 7-1) are implemented using workflow foundation technology and their interactions are modelled as closely as possible to an interoperability situation.

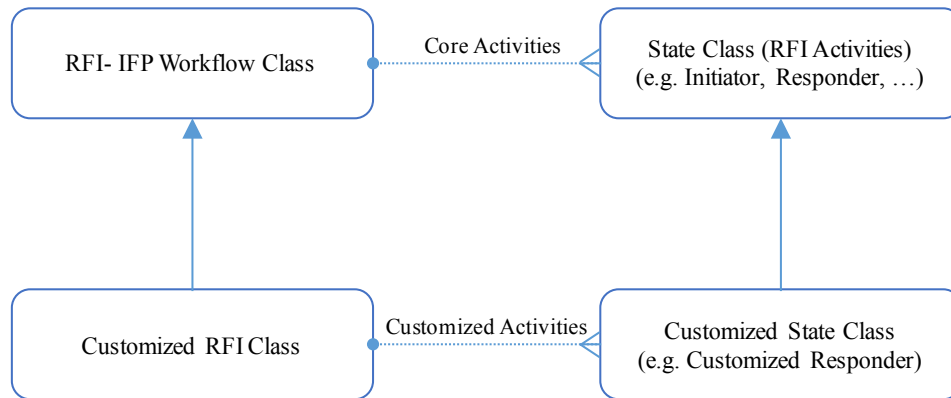


Figure 7-3: Modeling RFI Customized Workflows

The goal of this implementation is to validate the interaction and communication of two customized RFI workflow processes via their high-level public view as RFI-IFP processes, with the following purposes:

- Initiating an RFI process by, and in the middle of the execution of, another RFI process,
- Establishing a connection and exchanging the high-level (IFP) outline of processes, and other essential information,
- Maintain communication between processes through exchange of messages,
- Communicate state changes and status updates between activities,
- Closing the initiated activity after providing the requested information to the initiating process, and
- Using the high-level IFP structure as the means for all the interactions.

To establish the connection between workflow processes and initiate communication, Windows Communication Framework (WCF) has been used. WCF is the infrastructure for sending and

receiving messages between different parties. Messages are defined as general-purpose containers of data. The WCF object model supports sending messages using different data transfer protocols and enables technical and information interoperability. Adding messaging activities with message classes to workflow processes enable them to send and receive WCF messages (“Messaging Activities,” 2016).

The following messaging activities in WCF are used to establish interaction and communication between RFI workflow processes (“Messaging Activities,” 2016; White, 2013):

- InitializeCorrelation: establishes a correlation between messages prior to sending or receiving them. Usually, correlation is initialized when sending or receiving a message.
- Send: Sends a message to a service.
- SendReply: Sends a message to a service and anticipates receiving a response.
- Receive: Receives an incoming message.
- ReceiveReply: Receives an incoming message and send a reply back.

The information that is exchanged between workflow processes through messaging activities can generally be categorized into data fields, metadata fields, and attached documents. The IFP interoperability model facilitates the exchange of information between workflow processes via messages to provide the right people with the required information at the right time.

Exchange Record		
Data Fields	Meta-Data Fields	Attached Documents
Include process specification, process technical, and project data	Include process and document meta-data	Include primary and markup files

Figure 7-4: An Overall Exchange Record

The information exchange is conducted via an exchange record, and is comprised of data fields, metadata fields, and documents (Figure 7-4). Each component is defined as a data class, e.g. *Process Specification* Data class, and *Process Technical* Data Class. Customized workflow processes that inherit from, and are in conformance with, IFP processes can exchange information

seamlessly because the data classes share common fields. Figure 7-5 presents an example of data objects of an exchange record, and Figure 7-6 shows an example of message exchange between two RFI workflow processes.

<pre> Abstract Class ProjeData { Project Name Project ID Project Phase Discipline Company Department Facility Unit } Abstract Class ProcessSpecificationData { Process ID Title Description Originating Company Initiator Requested by Request date Priority Due Date Recipient Company Responder Response date Response Response Comment Remarks Actions Automated Transmittal Receipt Return Code External Approval Tracking Visibility } </pre>	<pre> Abstract Class ProcessTechnicalData { 3rd Party Reference No. Distribution Matrices Responsibility Matrices Check in and check out Searching Internal workflow Audit Ability To-do Lists Checklists Transmittal preparation Document due-date tracking Overdue document reports Supplier document indexes Previous transmittal response history Relationships -- External linked data } Abstract Class Document Metadata { Document ID Document Type File format Title File Name Issue Date Issue Purpose Revision ID Document Size Confidential? Regulatory? Allocated Doc. Number Markup Date Document Status Document Update Date Final Response Approval Status Approver Approval Date Page Count Summary } </pre>
--	--

Figure 7-5: Examples of Data Objects of an Exchange Record

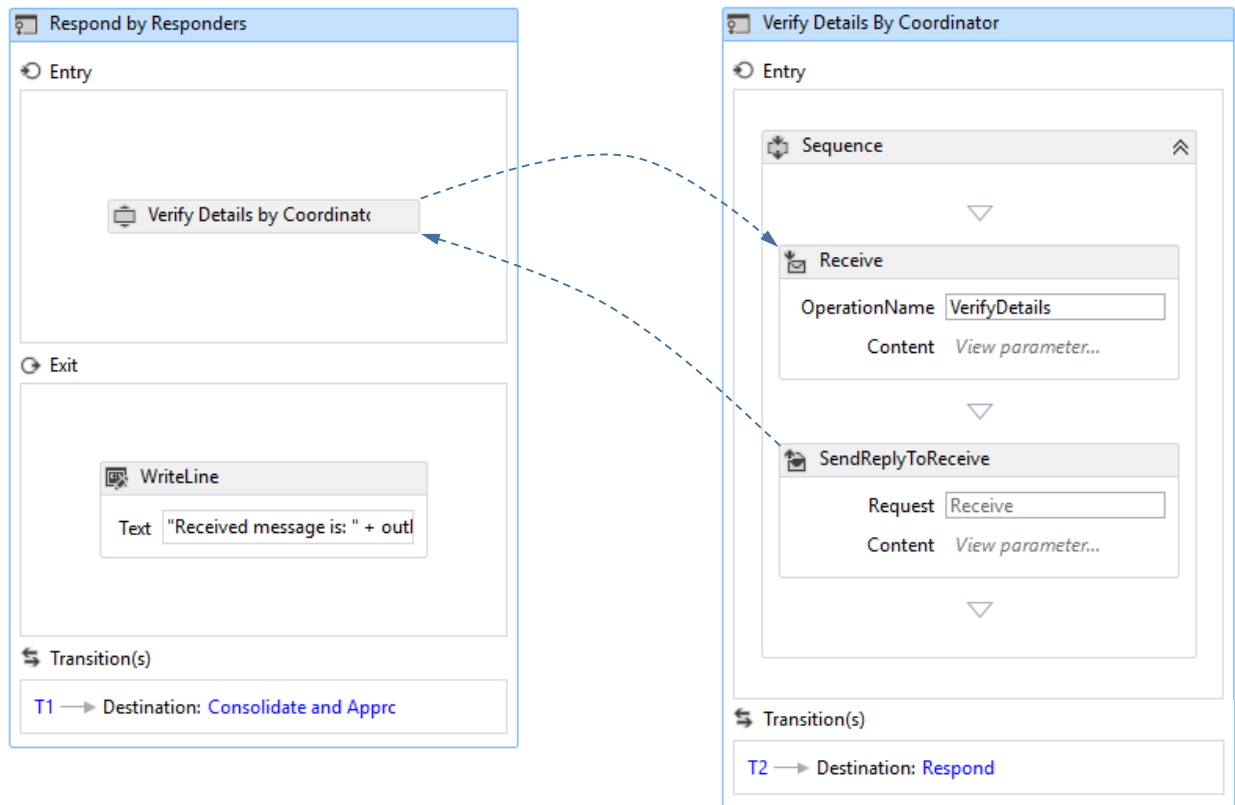


Figure 7-6: A Snapshot of Message Exchange Between Activities of Two RFI Processes

7.6 Discussion

To improve interoperability, the degree of interoperability among interoperable systems needs to be evaluated. This requires interoperability to be measured, and particular metrics need to be defined for this purpose. Although some research studies have been performed to deal with interoperability measurement and to define particular criteria for evaluating the degree of interoperability, the approaches mainly focus on development of different types of maturity models to evaluate the degree of interoperability. Developing interoperability measurement metrics is becoming an important challenge, due to the difficulty of identifying the attributes to characterize effective interoperability (Chen, Vallespir, et al., 2008).

One approach for evaluating the degree of interoperability is to categorize interoperability measures. Three types of interoperability measurement are identified: (1) interoperability potentiality measure, which evaluates the key attributes of a system, and its conformance with standard models and practices to assess the potential of the system to interoperate with any other

system; (2) interoperability compatibility measure, which is performed in the design or reengineering stage of a system, identifies barriers and evaluates the compatibility of two systems to exchange information; and (3) interoperability performance measure, which is performed during the test or operation phase of interoperable systems (Chen, Vallespir, et al., 2008).

To justify the role of the IFP system in improving process interoperability, this study presented the benefits of the IFP system and IFP interoperability model, and functionally demonstrated a basic implementation of interoperability for validation of the interoperability property of the IFP system. The IFP interoperability implementation is aligned with the interoperability potentiality and compatibility measures, but is not associated with the interoperability performance measure, because full-scale implementation of IFP processes is beyond the scope of this research.

Chapter 8

Conclusions and Future work

8.1 Summary and Conclusions

This study introduces the novel theory of Industry Foundation Processes (IFP) modeling system and offers an ontology and framework for its development and application. The IFP processes are defined as structured processes with the essence of industry best practices, possessing particular features, such as core structure, abstraction level, and inheritance rules that enable them to systematically be expanded to more complex processes tailored for specific types and conditions of construction projects. Explicit workflow inheritance rules allow methodical customization of IFP processes, and enable automated conformance checking of any workflow with its associated IFP process.

This study discusses the workflow inheritance concept and compares it with the traditional programming inheritance concept. It clarifies that they are different, and both are necessary for implementation of the IFP system. A prototype example of an IFP for the Request for Information (RFI) process – a commonly used process in the construction industry – was developed, using the C# programming language and Microsoft Workflow Foundation technology, to demonstrate the concept of an IFP system. The concept and methodology introduced, however, can be applied to any other common process in the construction industry, such as risk management, contract management, quality management, lessons learned, and processes in other domains.

In addition, automated conformance checking of any workflow with its associated IFP, based on the workflow inheritance rules, has been addressed in detail by developing an algorithm in a first-order logic language. Alloy, a structural modelling language based on first-order logic, is used to compare a customized version of a workflow with its associated IFP. The XAML file of the developed workflow in Visual Studio environment contains the structure of the workflow. This structure is transformed into the format accepted by Alloy to automate the conformance checking process directly from the workflow development environment.

Moreover, the core structure offered by the IFP system for common workflow processes in the construction industry, the workflow inheritance rules, and the conformance of customized workflow processes with the IFP processes are the basis for the IFP process interoperability model. The IFP interoperability model defines the external view of customized processes that are in conformance with an IFP to be the IFP process, and thus the exchange of information can be performed via interfaces that are defined between steps of the IFP process.

8.2 Contributions

The contributions of this study are summarized in seven main areas: (1) Developing theory of the IFP modeling system, (2) formalizing development approaches, (3) defining an ontology for the IFP processes, (4) validating the functionality of the system via implementation of the RFI process, (5) developing a workflow conformance checking algorithm, (6) developing an automated workflow conformance checking tool, and (7) proposing an IFP interoperability model. A brief description of these contributions is discussed in this section:

1. IFP System Theory – proposes the concept of foundation processes containing the essence of best practices, and how customized workflow implementations, for specific corporate and project situations, can be derived from the foundation processes analogous to the way that classes inherit properties from base classes.
2. Development Approaches – formalize the methods and mechanisms that can be used to transform best practices into structured workflow process, in such a way that the essence of the best practices is retained.
3. IFP Ontology – defines the required components of the IFP system, and establishes a framework for inheritance and customization of IFPs for specific corporate and project circumstances.
4. Deployment of the IFP system – investigates the applicability and usefulness of the IFP concept by customizing and implementing an IFP process in a workflow management system.

5. Workflow conformance checking algorithm – offers a novel methodology for analyzing and comparing the structure of two workflow processes. A first-order logic programming language was used for implementation of the algorithm.
6. Automated workflow conformance checking tool – is a combination of two Java-based applications to automate the process of workflow conformance checking from process design to visualization of the conformance checking results.
7. IFP interoperability model – proposes an interoperability model based on the IFP system to facilitate process interoperability between workflow processes that are in conformance with the IFP system.

8.3 Limitations

Despite the potential benefits of the industry foundation processes for the modeling of construction industry processes, the methodology proposed in this study has particular limitations:

- Although the theory and the application of the IFP modeling system has been validated by deployment of IFP processes, and by functional demonstration of the potential value through conformance and interoperability benefits in this study, a full scale validation via implementation of the IFP system in one or a few real projects has not been performed yet. Such a full scale implementation and validation would be a better examination and evaluation for the practical benefits of the IFP system in the construction industry.
- In the construction industry, workflows are often executed in a distributed setting. The prototype implementation presented in this study has been developed using Microsoft Workflow Foundation technology that fully supports distributed systems, but it has been implemented as a desktop application. Since Workflow Foundation facilitates separation of process design and process enactment from the type of application, and because the same classes that are typically used in distributed systems have been used in this desktop application, the implemented system can be considered an impartial validation for implementation of the RFI process. However, it is still a limitation of this study which can be addressed in a future work by developing a web-based distributed system.

- The IFP system and its ontology components have been defined based on careful investigation and analysis of several process implementations, and consultation with industry experts; however, having access to and analyzing process implementations in more projects might lead to some updates on the definition of components or details.

8.4 Recommendations for Future Work

Introducing the theory, application, and potential value of the IFP system is expected to open new research initiatives to enhance process conformance and improve process interoperability in the domain of the construction industry. The following recommendations for future research are proposed based on this thesis:

- The inheritance rules that have been used for workflow conformance checking in this study are strict rules that do not allow skipping any of the core activities or changing the sequence of them. As a future work these rules can be relaxed to some extent, *i.e.* to allow change in the sequence of particular core activities, or to allow skipping particular core activities in certain situations, and investigating how these changes affect the conformance checking algorithm.
- To validate the functionality and benefits of the IFP process modeling system it would be ideal to use the IFP system for a set of IFP processes in one or more construction projects. However, such full deployment of the IFP system to fulfill the requirements of a real construction project is a complex task and beyond the scope of this research. In addition, application of such a system in an existing project as a case study requires several types of permits and numerous resources, which would not be feasible as part of this research.
- While industry partners and experts consulted in this research process highly value the process modeling system offered by the IFP system, and its application and benefits have been validated by functional demonstration, the effect of improved conformance and interoperability has not been evaluated with any survey or metrics on the project performance. Future research that would compare its deployment on a large set of mega-projects with current workflow management and implementations protocols would be worthwhile for also validating its impact on project performance.

- The automated conformance checking algorithm and tool provides a backward conformance checking approach, in which the structure of an implemented workflow process is analyzed against the IFP, to discover non-conformant behavior. Integrating inheritance rules and conformance checking algorithm into process design tools, to facilitate forward conformance checking by notifying the developer of a non-conformance behavior, in the process design stage, is also considered a future work.

REFERENCES

- Aalst, W. M. P. van der. (2004). Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig, & G. Rozenberg (Eds.), *Lectures on Concurrency and Petri Nets* (pp. 1–65). Springer Berlin Heidelberg. Retrieved from http://link.springer.com.proxy.lib.uwaterloo.ca/chapter/10.1007/978-3-540-27755-2_1
- Al Qady, M., & Kandil, A. (2013). Document Discourse for Managing Construction Project Documents. *Journal of Computing in Civil Engineering*, 27(5), 466–475. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0000201](https://doi.org/10.1061/(ASCE)CP.1943-5487.0000201)
- Anand, P., & Singh, M. D. (2011). Understanding the Knowledge Management. *International Journal of Engineering Science and Technology*, 3(2), 926–939.
- Basten, T., & van der Aalst, W. M. P. (2001). Inheritance of behavior. *The Journal of Logic and Algebraic Programming*, 47(2), 47–145. [https://doi.org/10.1016/S1567-8326\(00\)00004-7](https://doi.org/10.1016/S1567-8326(00)00004-7)
- Benchmarking & Metrics Implementation Toolkit*. (2004). Construction Industry Institute.
- Botha, A., Kourie, D., & Snyman, R. (2008). *Coping with Continuous Change in the Business Environment: Knowledge Management and Knowledge Management Technology* (1 edition). Oxford, UK: Chandos Publishing.
- Caldas, C., Soibelman, L., & Gasser, L. (2005). Methodology for the Integration of Project Documents in Model-Based Information Systems. *Journal of Computing in Civil Engineering*, 19(1), 25–33. [https://doi.org/10.1061/\(ASCE\)0887-3801\(2005\)19:1\(25\)](https://doi.org/10.1061/(ASCE)0887-3801(2005)19:1(25))
- Cardoso, J., Bostrom, R. P., & Sheth, A. (2004). Workflow Management Systems and ERP Systems: Differences, Commonalities, and Applications. *Information Technology and Management*, 5(3–4), 319–338. <https://doi.org/10.1023/B:ITEM.0000031584.14039.99>

- Chanmeka, A., Thomas, S. R., Caldas, C. H., & Mulva, S. P. (2012). Assessing key factors impacting the performance and productivity of oil and gas projects in Alberta. *Canadian Journal of Civil Engineering*, 39(3), 259–270. <https://doi.org/10.1139/l11-128>
- Chen, D., Doumeingts, G., & Vernadat, F. (2008). Architectures for enterprise integration and interoperability: Past, present and future. *Computers in Industry*, 59(7), 647–659. <https://doi.org/10.1016/j.compind.2007.12.016>
- Chen, D., Vallespir, B., Daclin, N., & others. (2008). An Approach for Enterprise Interoperability Measurement. In *MoDISE-EUS* (pp. 1–12). Retrieved from <http://ceur-ws.org/Vol-341/paper1.pdf>
- Chung, B., Skibniewski, M., & Kwak, Y. (2009). Developing ERP Systems Success Model for the Construction Industry. *Journal of Construction Engineering and Management*, 135(3), 207–216. [https://doi.org/10.1061/\(ASCE\)0733-9364\(2009\)135:3\(207\)](https://doi.org/10.1061/(ASCE)0733-9364(2009)135:3(207))
- Construction Industry Institute. (2015, August). *2015 CII Annual Conference*. Boston, Massachusetts. Retrieved from <https://www.construction-institute.org/ac/2015/index.cfm>
- Cook, S. A. (1971). The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151–158). New York, NY, USA: ACM. <https://doi.org/10.1145/800157.805047>
- Dennis, G. D. (Gregory D. (2009). *A relational framework for bounded program verification* (Thesis). Massachusetts Institute of Technology. Retrieved from <http://dspace.mit.edu/handle/1721.1/55097>
- Eén, N., & Sörensson, N. (2004). An Extensible SAT-solver. In E. Giunchiglia & A. Tacchella (Eds.), *Theory and Applications of Satisfiability Testing* (pp. 502–518). Springer Berlin

- Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-24605-3_37
- El-Gohary, N., & El-Diraby, T. (2010). Dynamic Knowledge-Based Process Integration Portal for Collaborative Construction. *Journal of Construction Engineering and Management*, *136*(3), 316–328. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0000147](https://doi.org/10.1061/(ASCE)CO.1943-7862.0000147)
- El-Mashaleh, M., O'Brien, W., & Minchin, R. (2006). Firm Performance and Information Technology Utilization in the Construction Industry. *Journal of Construction Engineering and Management*, *132*(5), 499–507. [https://doi.org/10.1061/\(ASCE\)0733-9364\(2006\)132:5\(499\)](https://doi.org/10.1061/(ASCE)0733-9364(2006)132:5(499))
- Faust, B. (2007). Implementation of tacit knowledge preservation and transfer methods. In *International Conference on Knowledge Management in Nuclear Facilities*. Retrieved from <http://www.fraserhealth.ca/media/Implementation-of-Tacit-Knowledge-Presevation-and-Transfer-Methods.pdf>
- Froese, T. (2003). Future directions for IFC-based interoperability. Retrieved March 7, 2013, from http://www.itcon.org/cgi-bin/works/Show?2003_17
- GCR, N. (2004). Cost analysis of inadequate interoperability in the US capital facilities industry. Retrieved from http://www.bentleyuser.dk/sites/default/files/nist_report.pdf
- Georgakopoulos, D., Hornick, M., & Sheth, A. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, *3*(2), 119–153. <https://doi.org/10.1007/BF01277643>
- Georgiadis, L., Tarjan, R. E., & Werneck, R. F. (2006). Finding Dominators in Practice. *Journal of Graph Algorithms and Applications*, *10*(1), 69–94. <https://doi.org/10.7155/jgaa.00119>

- Ghosh, S., Negahban, S., Kwak, Y. H., & Skibniewski, M. J. (2011). Impact of sustainability on integration and interoperability between BIM and ERP - A governance framework. In *Technology Management Conference (ITMC), 2011 IEEE International* (pp. 187–193). <https://doi.org/10.1109/ITMC.2011.5995975>
- Giaglis, G. M. (2001). A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques. *International Journal of Flexible Manufacturing Systems*, 13(2), 209–228. <https://doi.org/10.1023/A:1011139719773>
- Gibbons, P., Arzt, N., Burke-Beebe, S., Chute, C., Dickinson, G., Flewelling, T., ... others. (2007). Coming to terms: Scoping interoperability for health care. Retrieved from <http://www.citeulike.org/group/15536/article/10705562>
- Golzarpoor, B., Haas, C. T., & Rayside, D. (2016). Improving process conformance with Industry Foundation Processes (IFP). *Advanced Engineering Informatics*, 30(2), 143–156. <https://doi.org/10.1016/j.aei.2016.02.005>
- Governatori, G., & Sadiq, S. (2009). The journey to business process compliance. *Handbook of Research on BPM*, 426–454.
- Hollingsworth, D. (1995). Workflow Management Coalition - The Workflow Reference Model. IIBA. (2015). *A Guide to the Business Analysis Body of Knowledge* (3rd ed. edition). Lightning Source Inc.
- IT Catalysts. (2013). Retrieved November 19, 2014, from <http://www.itcatalysts.com>
- Jackson, D. (2002). Micromodels of software: lightweight modelling and analysis with Alloy. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/239599105_Micromodels_of_software_lightweight_modelling_and_analysis_with_Alloy

- Jackson, D. (2011). *Software Abstractions: Logic, Language, and Analysis* (revised edition edition). Cambridge, Mass: The MIT Press.
- Kang, Y., O'Brien, W., Dai, J., Mulva, S., Thomas, S., Chapman, R., & Butry, D. (2013). Interaction Effects of Information Technologies and Best Practices on Construction Project Performance. *Journal of Construction Engineering and Management*, 139(4), 361–371. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0000627](https://doi.org/10.1061/(ASCE)CO.1943-7862.0000627)
- Kang, Y., O'Brien, W. J., & Mulva, S. P. (2013). Value of IT: Indirect impact of IT on construction project performance via Best Practices. *Automation in Construction*, 35, 383–396. <https://doi.org/10.1016/j.autcon.2013.05.011>
- Kang, Y., O'Brien, W. J., & O'Connor, J. T. (2012). Analysis of information integration benefit drivers and implementation hindrances. *Automation in Construction*, 22, 277–289. <https://doi.org/10.1016/j.autcon.2011.09.003>
- Kang, Y., O'Brien, W., Thomas, S., & Chapman, R. (2008). Impact of Information Technologies on Performance: Cross Study Comparison. *Journal of Construction Engineering and Management*, 134(11), 852–863. [https://doi.org/10.1061/\(ASCE\)0733-9364\(2008\)134:11\(852\)](https://doi.org/10.1061/(ASCE)0733-9364(2008)134:11(852))
- Khan, W. A., Hussain, M., Latif, K., Afzal, M., Ahmad, F., & Lee, S. (2013). Process interoperability in healthcare systems with dynamic semantic web services. *Computing*. <https://doi.org/10.1007/s00607-012-0239-3>
- Krishnakumar, N., & Sheth, A. (1995). Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2), 155–186. <https://doi.org/10.1007/BF01277644>

- Kubicek, H., & Cimander, R. (2009). Three dimensions of organizational interoperability. *European Journal of ePractice*, 6. Retrieved from http://www.ifib-consult.de/publikationsdateien/Kubicek_Cimander_ePractice_Journal_vol_6.pdf
- Lee, L. L. (2005). Balancing business process with business practice for organizational advantage. *Journal of Knowledge Management*, 9(1), 29–41.
<https://doi.org/10.1108/13673270510582947>
- Lee, S., Thomas, S., Macken, C., Chapman, R., Tucker, R., & Kim, I. (2005). Economic Value of Combined Best Practice Use. *Journal of Management in Engineering*, 21(3), 118–124.
[https://doi.org/10.1061/\(ASCE\)0742-597X\(2005\)21:3\(118\)](https://doi.org/10.1061/(ASCE)0742-597X(2005)21:3(118))
- Lengauer, T., & Tarjan, R. E. (1979). A Fast Algorithm for Finding Dominators in a Flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1), 121–141.
<https://doi.org/10.1145/357062.357071>
- Lewis, G. A. (2013). Role of standards in cloud-computing interoperability. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on* (pp. 1652–1661). IEEE.
Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6480040
- Ly, L. T., Maggi, F. M., Montali, M., Rinderle-Ma, S., & van der Aalst, W. M. P. (2015). Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information Systems*, 54, 209–234. <https://doi.org/10.1016/j.is.2015.02.007>
- Malone, T. W., Crowston, K., & Herman, G. A. (2003). *Organizing Business Knowledge: The MIT Process Handbook*. Cambridge, Mass: The MIT Press.
- Mannhardt, F., Leoni, M. de, Reijers, H. A., & Aalst, W. M. P. van der. (2015). Balanced multi-perspective checking of process conformance. *Computing*, 1–31.
<https://doi.org/10.1007/s00607-015-0441-1>

- Messaging Activities. (2016, April 21). Retrieved April 21, 2016, from [https://msdn.microsoft.com/en-us/library/ee358739\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ee358739(v=vs.110).aspx)
- Microsoft Developer Network. (2015a, April 21). A Developer's Introduction to Windows Workflow Foundation (WF) in .NET 4. Retrieved April 21, 2015, from <https://msdn.microsoft.com/en-us/library/ee342461.aspx>
- Microsoft Developer Network. (2015b, April 21). The Workflow Way: Understanding Windows Workflow Foundation. Retrieved April 21, 2015, from <https://msdn.microsoft.com/en-us/library/dd851337.aspx>
- Mühlen, M. zur, & Shapiro, R. (2010). Business Process Analytics. In J. vom Brocke & M. Rosemann (Eds.), *Handbook on Business Process Management 2* (pp. 137–157). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01982-1_7
- O'Connor, J. T., & Dodd, S. C. (2000). Achieving integration on capital projects with enterprise resource planning systems. *Automation in Construction*, 9(5–6), 515–524. [https://doi.org/10.1016/S0926-5805\(00\)00062-5](https://doi.org/10.1016/S0926-5805(00)00062-5)
- Project Change Management - Special Publication 43-1*. (1994). Construction Industry Institute (CII).
- Project Management Institute. (2013). *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)–Fifth Edition* (5 edition). Newtown Square, Pennsylvania: Project Management Institute.
- Prosser, R. T. (1959). Applications of Boolean Matrices to the Analysis of Flow Diagrams. In *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference* (pp. 133–138). New York, NY, USA: ACM. <https://doi.org/10.1145/1460299.1460314>

- Schwartz, J. T., Dewar, R. B., Schonberg, E., & Dubinsky, E. (1986). *Programming with Sets: an Introduction to SETL*. New York, NY, USA: Springer-Verlag New York, Inc.
- Shahi, A., Haas, C., West, J., & Akinci, B. (2014). Workflow-Based Construction Research Data Management and Dissemination. *Journal of Computing in Civil Engineering*, 28(2), 244–252. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0000251](https://doi.org/10.1061/(ASCE)CP.1943-5487.0000251)
- Shan, Y., Goodrum, P. M., Zhai, D., Haas, C., & Caldas, C. H. (2011). The impact of management practices on mechanical construction productivity. *Construction Management and Economics*, 29(3), 305–316. <https://doi.org/10.1080/01446193.2010.538070>
- Shen, W. (2010). Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review. *Advanced Engineering Informatics*, 24(2), 196–207.
- Shokri, S., Safa, M., Haas, C. T., Haas, R. C., Maloney, K., & MacGillivray, S. (2012). Interface management model for mega capital projects. In *Proc. of the 2012 Construction Research Congress, Purdue University, IN, United States* (pp. 447–456). Retrieved from <http://rebar.ecn.purdue.edu/crc2012/papers/pdfs/-242.pdf>
- Skibniewski, M., & Ghosh, S. (2009). Determination of Key Performance Indicators with Enterprise Resource Planning Systems in Engineering Construction Firms. *Journal of Construction Engineering and Management*, 135(10), 965–978. [https://doi.org/10.1061/\(ASCE\)0733-9364\(2009\)135:10\(965\)](https://doi.org/10.1061/(ASCE)0733-9364(2009)135:10(965))
- State Machine Workflows. (2015, April 21). Retrieved April 21, 2015, from [https://msdn.microsoft.com/en-us/library/ee264171\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ee264171(v=vs.110).aspx)

- Szykman, S., Fennes, S. J., Keirouz, W., & Shooter, S. B. (2001). A foundation for interoperability in next-generation product development systems. *Computer-Aided Design*, 33(7), 545–559. [https://doi.org/10.1016/S0010-4485\(01\)00053-7](https://doi.org/10.1016/S0010-4485(01)00053-7)
- Taghiabadi, E. R., Fahland, D., Dongen, B. F. van, & Aalst, W. M. P. van der. (2013). Diagnostic Information for Compliance Checking of Temporal Compliance Requirements. In C. Salinesi, M. C. Norrie, & Ó. Pastor (Eds.), *Advanced Information Systems Engineering* (pp. 304–320). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-38709-8_20
- Tang, P., & Akinci, B. (2012). Automatic execution of workflows on laser-scanned data for extracting bridge surveying goals. *Advanced Engineering Informatics*, 26(4), 889–903. <https://doi.org/10.1016/j.aei.2012.07.004>
- Tao Lue Wu. (2015). *Theoretical Foundation of Workflow Conformance* (Undergraduate Student Work Report). University of Waterloo.
- The Australian government business process interoperability framework. (2007). Retrieved November 7, 2016, from <http://trove.nla.gov.au/version/42816666>
- Thomas, S., Lee, S., Spencer, J., Tucker, R., & Chapman, R. (2004). Impacts of Design/Information Technology on Project Outcomes. *Journal of Construction Engineering and Management*, 130(4), 586–597. [https://doi.org/10.1061/\(ASCE\)0733-9364\(2004\)130:4\(586\)](https://doi.org/10.1061/(ASCE)0733-9364(2004)130:4(586))
- Ullmann, J. R. (1976). An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1), 31–42. <https://doi.org/10.1145/321921.321925>
- van der Aalst, W. M. (2002). Inheritance of dynamic behaviour in UML. *MOCA*, 2, 105–120.

- van der Aalst, W. M. ., & Basten, T. (2002). Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1–2), 125–203.
[https://doi.org/10.1016/S0304-3975\(00\)00321-2](https://doi.org/10.1016/S0304-3975(00)00321-2)
- Van Der Aalst, W. M. P. (2003). *Inheritance of business processes: A journey visiting four notorious problems* (Vol. 2472).
- van der Aalst, W. M. P. (2014). Business process management as the “Killer App” for Petri nets. *Software & Systems Modeling*. <https://doi.org/10.1007/s10270-014-0424-2>
- Veer, H., & Wiles, A. (2008, April). Achieving Technical Interoperability - the ETSI Approach - 3rd edition. European Telecommunications Standards Institute (ETSI).
- Weske, M. (2012). *Business process management: concepts, languages, architectures*. Berlin [etc.]: Springer.
- White, B. (2013). *Pro WF 4.5*. Berkeley, CA: Apress. Retrieved from
<http://link.springer.com/10.1007/978-1-4302-4384-7>
- Zhai, D., Goodrum, P., Haas, C., & Caldas, C. (2009). Relationship between Automation and Integration of Construction Information Systems and Labor Productivity. *Journal of Construction Engineering and Management*, 135(8), 746–753.
[https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0000024](https://doi.org/10.1061/(ASCE)CO.1943-7862.0000024)

Appendix A

Glossary of Terms

- Automation* – Utilization of electronic or computerized tools to make a task more efficient
- Best Practice* – A process or method that leads to superior results comparing to other means
- BIM (Building Information Modeling)* – The process of modeling buildings and infrastructures for planning, design, construction, and/or management purposes
- BPM (Business Process Management)* – An approach for monitoring and optimizing business processes within an organization
- BPMN (Business Process Model and Notation)* – A standard notation for modeling business processes
- CII (The Construction Industry Institute)* – A consortium of multiple companies, in the construction-related industries, working together to enhance business effectiveness and sustainability within the industry
- COAA (The Construction Owners Association of Alberta)* – An association that provides leadership to construction and industrial maintenance industries in Alberta
- Conformance* – The act of complying with a certain standards, guidelines, or specifications.
- EPPM system – Electronic Product and Process Management System* – A type of workflow management system that is used specifically for managing mega capital projects
- ERP (Enterprise Resource Planning)* – A business management tool that allows a company to manage its business activities
- IFC (Industry Foundation Classes)* – A neutral and open file format data model intended for describing, exchanging, and sharing of data within the building and construction industry.
- Inheritance* – A mechanism in programming that allows a class to inherit features of another class; it increases the reusability of system components
- Integration* – The ability of sharing information from multiple sources between two or more systems.
- Interoperability* – The ability to communicate and exchange information, within different information technology systems
- IT (Information Technology)* – The applications of computer-related systems used in the processing and distribution of data
- Mega-Project* – Large scale capital projects with substantial impacts, typically involving multiple stakeholders, and costing more than US1\$ billion

OOP (Object-Oriented Programming) - A programming language model that focuses on manipulating data objects rather than the logic required to manipulate the data

Ontology – A formal description or specification of all aspects of a topic

PMI (The Project Management Institute) – A professional membership association for the project, program, and portfolio management profession aimed to improve organizational success

Workflow Engine – A software application that governs enactment of processes based on predefined rules and specifications

Workflow Template – A predefined workflow that contains the most common activities and relationships

Workflow Management System (WfMS) – A software system for managing, monitoring, and executing workflow processes

Appendix B

Samples of Core WF Code for Deployment of RFI Process

ActivityViews.cs file

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
using System.Windows.Forms;

namespace RFIPProcessWorkflowActivities
{
    public class ResumeBookmarkObject
    {
        public string responder;
        public string consolidator;

        public ResumeBookmarkObject(string responder, string consolidator)
        {
            this.responder = responder;
            this.consolidator = consolidator;
        }
    };

    //starts the corresponding form
    public sealed class InvokeInitiatorView : CodeActivity {
        //get rfi id of the current workflow instance
        [RequiredArgument]
        public InArgument<int> RFI_id { get; set; }

        protected override void Execute(CodeActivityContext context)
        {
            //starts the initiator form
            Application.Run(new InitiatorView(RFI_id.Get(context)));
        }
    }

    public sealed class InvokeCoordinatorView : NativeActivity<string>
    {
        //get rfi id of the current workflow instance
        [RequiredArgument]
        public InArgument<int> RFI_id { get; set; }

        protected override void Execute(NativeActivityContext context)
        {
            CoordinatorView coordinatorView = new CoordinatorView(RFI_id.Get(context));
            coordinatorView.ShowDialog();
            Console.WriteLine("result coor = {0}\n", coordinatorView.result);
            this.Result.Set(context, coordinatorView.result);
        }
    }

    public sealed class InvokeInitiatorResponserView : NativeActivity<string>
    {
        //get rfi id of the current workflow instance
        [RequiredArgument]
```

```

    public InArgument<int> RFI_id { get; set; }

    protected override void Execute(NativeActivityContext context)
    {
        InitiatorResponseView initiatorResponseView = new
InitiatorResponseView(RFI_id.Get(context));
        initiatorResponseView.ShowDialog();
    }
}

//starts the corresponding form
public sealed class InvokeConsolidatorView : NativeActivity<string> {
    //get rfi id of the current workflow instance
    [RequiredArgument]
    public InArgument<int> RFI_id { get; set; }
    public InArgument<string> responder { get; set; }

    protected override void Execute(NativeActivityContext context)
    {
        ConsolidatorView consolidatorView = new ConsolidatorView(RFI_id.Get(context),
responder.Get(context));
        consolidatorView.ShowDialog();
        Console.WriteLine("result cons = {0}\n", consolidatorView.result);
        this.Result.Set(context, consolidatorView.result);
    }
}

//starts the corresponding form
public sealed class InvokeResponderView : NativeActivity<string> {
    //get rfi id of the current workflow instance
    [RequiredArgument]
    public InArgument<int> RFI_id { get; set; }
    public InArgument<string> Responder { get; set; }

    protected override void Execute(NativeActivityContext context)
    {
        Console.WriteLine("Invoking responderview: {0}", Responder.Get(context));
        ResponderView responderView = new ResponderView(RFI_id.Get(context),
Responder.Get(context));
        responderView.ShowDialog();
        Console.WriteLine("result resp = {0}\n", responderView.result);
        this.Result.Set(context, responderView.result);
    }
}

public sealed class transitionView : NativeActivity
{
    [RequiredArgument]
    public InArgument<string> BookmarkName { get; set; }

    public OutArgument<string> responder { get; set; }
    public OutArgument<string> consolidator { get; set; }

    protected override void Execute(NativeActivityContext context)
    {
        string name = BookmarkName.Get(context);
        Console.WriteLine("Creating Bookmark {0}", name);
        context.CreateBookmark(name, new BookmarkCallback(OnReadComplete));
    }
}

```

```

    }

    protected override bool CanInduceIdle
    {
        get { return true; }
    }

    void OnReadComplete(NativeActivityContext context, Bookmark bookmark, object state)
    {
        Console.WriteLine("OnReadComplete resp is = {0}",
            ((ResumeBookmarkObject)state).responder.ToString());
        Console.WriteLine("OnReadComplete cons is = {0}",
            ((ResumeBookmarkObject)state).consolidator.ToString());

        this.responder.Set(context, ((ResumeBookmarkObject)state).responder.ToString());
        this.consolidator.Set(context,
            ((ResumeBookmarkObject)state).consolidator.ToString());
        Console.WriteLine("Resuming bookmark");
    }
}
}
}

```

InitiatorView.cs file

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace RFIProcessWorkflowActivities
{
    public partial class InitiatorView : Form
    {
        private InitiatorViewObject initObj;
        private bool submitClicked = false;

        public InitiatorView(int rfi_id)
        {
            Console.WriteLine("in init view: {0}", rfi_id);
            // constructor will either load new rfi if rfi_id
            //does not exist in db, orwill load one from the db if one does
            initObj = new InitiatorViewObject(rfi_id);
            InitializeComponent();

            // these will always be there
            this.rfi_IDTextBox.Text = initObj.RFI_ID.ToString();
            this.createdDatePicker.Value = initObj.DateCreated;
            this.initiatorTextBox.Text = initObj.Initiator;
            this.statusTextBox.Text = initObj.Status;

            this.infoRequestedTextBox.Text = initObj.InfoRequested;
            this.reasonTextBox.Text = initObj.Reason;
            this.project_IDTextbox.Text = initObj.Project_ID;
            this.titleTextBox.Text = initObj.ProjectTitle;

```



```

        if (initObj.RequiredByDate.HasValue) {
            this.requiredByDatePicker.Value = initObj.RequiredByDate.Value;
        }
        this.project_IDTextbox.Text = "Construction Project 3";
    }

private void SubmitButton_Click(object sender, EventArgs e)
{
    submitClicked = true;
    retrieveViewValues();
    if (initObj.IsIncomplete)
    {
        submitClicked = false;
        MessageBox.Show("Form is incomplete", "User Error");
        return;
    }
    else
    {
        if (recordExists()) {
            updateDB();
        }
        else {
            insertIntoDB();
        }
        this.Close();
    }
}

private void insertIntoDB()
{
    // insert a RFI_ID one higher
    SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);
    SqlCommand insertCommand = new SqlCommand("INSERT INTO
RFI_Submission_Table(RFI_ID, InfoRequested, Reason, DateCreated, Coordinator, Initiator,
ProjectTitle, Project_ID, Status, RequiredByDate) VALUES(@rf, @ir, @re, @dc, @co, @in, @pt,
@pi, @st, @rd)", con);
    con.Open();
    //@rf, @ir, @re, @dc, @co, @in, @pt, @pi, @st, @rd
    insertCommand.Parameters.AddWithValue("@rf", initObj.RFI_ID);
    insertCommand.Parameters.AddWithValue("@ir", initObj.InfoRequested);
    insertCommand.Parameters.AddWithValue("@re", initObj.Reason);
    insertCommand.Parameters.AddWithValue("@dc", initObj.DateCreated);
    insertCommand.Parameters.AddWithValue("@co", "Coordinator1");
    insertCommand.Parameters.AddWithValue("@in", initObj.Initiator);
    insertCommand.Parameters.AddWithValue("@pt", initObj.ProjectTitle);
    insertCommand.Parameters.AddWithValue("@pi", initObj.Project_ID);
    insertCommand.Parameters.AddWithValue("@st", "Awaiting Coordination");
    insertCommand.Parameters.AddWithValue("@rd", initObj.RequiredByDate);
    insertCommand.ExecuteNonQuery();
    con.Close();
}

private void updateDB()
{
    SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);

```

```

        SqlCommand insertCommand = new SqlCommand("UPDATE RFI_Submission_Table SET
InfoRequested = @ir, Reason=@re, ProjectTitle=@pt, Project_ID=@pi, Status=@st,
RequiredByDate=@rd WHERE RFI_ID=@rf", con);

        //@rf, @ir, @re, @dc, @co, @in, @pt, @pi, @st, @rd
insertCommand.Parameters.AddWithValue("@ir", initObj.InfoRequested);
insertCommand.Parameters.AddWithValue("@re", initObj.Reason);
insertCommand.Parameters.AddWithValue("@pt", initObj.ProjectTitle);
insertCommand.Parameters.AddWithValue("@pi", initObj.Project_ID);
insertCommand.Parameters.AddWithValue("@st", "Awaiting Coordination");
insertCommand.Parameters.AddWithValue("@rd", initObj.RequiredByDate);
insertCommand.Parameters.AddWithValue("@rf", initObj.RFI_ID);

        con.Open();
insertCommand.ExecuteNonQuery();
con.Close();
}
private void InitiatorView_FormClosing(object sender, FormClosingEventArgs e)
{
    if (!submitClicked)
    {
        DialogResult dr = MessageBox.Show("You are about to close without submitting.
Are you sure you want to close?", "Cancelling", MessageBoxButtons.YesNo);
        if(dr == DialogResult.Yes)
        {
            submitClicked = false;
            removeRFIIInstance();
        }
        else
        {
            e.Cancel = true;
        }
    }
}

private void retrieveViewValues()
{
    initObj.Project_ID = this.project_IDTextbox.Text;
    initObj.ProjectTitle = this.titleTextbox.Text;
    initObj.InfoRequested = this.infoRequestedTextBox.Text;
    initObj.Reason = this.reasonTextbox.Text;
    initObj.RequiredByDate = this.requiredByDatePicker.Value;
}
private void removeRFIIInstance()
{
    SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);
    SqlCommand deleteCommand1 = new SqlCommand("DELETE FROM RFI_Instance WHERE RFI_ID
= @r", con);
    SqlCommand deleteCommand2 = new SqlCommand("DELETE FROM RFI_Submission_Table WHERE
RFI_ID = @r", con);

    con.Open();
deleteCommand1.Parameters.AddWithValue("@r", initObj.RFI_ID);
deleteCommand2.Parameters.AddWithValue("@r", initObj.RFI_ID);

    deleteCommand1.ExecuteNonQuery();
deleteCommand2.ExecuteNonQuery();
}

```

```

        con.Close();
    }

    private bool recordExists()
    {
        bool ret;
        SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConne
ctionString);
        SqlCommand selectCommand = new SqlCommand("SELECT * FROM RFI_Submission_Table
WHERE RFI_ID = (" + this.initObj.RFI_ID + ")", con);
        con.Open();
        SqlDataReader d = selectCommand.ExecuteReader();
        d.Read();
        ret = d.HasRows;
        d.Close();
        con.Close();
        return ret;
    }
}
}
}

```

CoordinatorView.cs file

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Activities;
using System.Data.SqlClient;

namespace RFIProcessWorkflowActivities
{
    public partial class CoordinatorView : Form
    {
        private CoordinatorViewObject coorObj;
        public string result = "cancelled";

        public CoordinatorView(int rfi_id)
        {
            coorObj = new CoordinatorViewObject(rfi_id);
            InitializeComponent();

            // values that cannot be changed by the user
            this.rfi_IDTextBox.Text = coorObj.RFI_ID.ToString();
            this.statusTextBox.Text = coorObj.Status;
            this.createdDatePicker.Value = coorObj.DateCreated;
            this.initiatorTextBox.Text = coorObj.Initiator;

            // These values are from the Initiator
            this.project_IDTextBox.Text = coorObj.Project_ID;
            this.titleTextBox.Text = coorObj.ProjectTitle;
            this.inforRequestedTextBox.Text = coorObj.InfoRequested;
            this.reasonTextBox.Text = coorObj.Reason;

```

```

        this.requiredByDatePicker.Value = (DateTime)coorObj.RequiredByDate; // nullable,
must cast
    }

    private void updateDB()
    {
        SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConne
ctionString);
        con.Open();
        // insert into Submission Table
        SqlCommand insertCommand;
        //get the highest RFI_ID
        SqlCommand selectCommand;
        // insert into Response Table
        SqlCommand insertCommand2;
        // insert into responder Table
        SqlCommand insertCommand3;

        // update for responder
        if (result == "accepted") {
            // update submission table
            insertCommand = new SqlCommand("UPDATE RFI_Submission_Table SET Status =
'Awaiting Response' WHERE RFI_ID = (" + coorObj.RFI_ID + ")", con);
            insertCommand.ExecuteNonQuery();

            // get highest response_ID from table
            selectCommand = new SqlCommand("SELECT MAX(Response_ID) FROM
RFI_Response_Table", con);
            SqlDataReader d = selectCommand.ExecuteReader();
            d.Read();
            // to avoid confusion with RFI_ID the Response_ID will start at 100
            int response_ID = d[0] != DBNull.Value ? Convert.ToInt32(d[0])+1 : 101;
            d.Close();

            // update Response table with the response_ID
            insertCommand2 = new SqlCommand("INSERT INTO RFI_Response_Table(RFI_ID,
Response_ID) VALUES(@rf, @reid)", con);
            insertCommand2.Parameters.AddWithValue("@rf", coorObj.RFI_ID);
            insertCommand2.Parameters.AddWithValue("@reid", response_ID);
            insertCommand2.ExecuteNonQuery();

            // add each repsonder to the Responder Table
            // the responses at this state will be 'awaiting answer'
            insertCommand3 = new SqlCommand("INSERT INTO Responder_Table(Response_ID,
Responder, ResponseApproved) VALUES(@reid, @resp, @ra)", con);
            insertCommand3.Parameters.AddWithValue("@reid", response_ID);
            insertCommand3.Parameters.AddWithValue("@ra", "awaiting answer");

            foreach (string responder in responderListBox.SelectedItems)
            {
                Console.WriteLine("Foreach loop : {0}", responder);
                insertCommand3.Parameters.AddWithValue("@resp", responder);
                insertCommand3.ExecuteNonQuery();
            }
        }
        // update for initiator
        else if (result == "rejected") {

```

```

        insertCommand = new SqlCommand("UPDATE RFI_Submission_Table SET Comment = '" +
        coorObj.Comments + "', Status = 'Awaiting Initiation' WHERE RFI_ID = (" + coorObj.RFI_ID +
        ")", con);
        insertCommand.ExecuteNonQuery();
    }

    con.Close();
}

private void retrieveViewValues()
{
    coorObj.Comments = this.commentsTextBox.Text;
}

public void acceptClick(object sender, EventArgs e) {
    result = "accepted";
    SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);
    con.Open();
    // insert into Submission Table
    SqlCommand insertCommand;
    //get the highest RFI_ID
    SqlCommand selectCommand;
    // insert into Response Table
    SqlCommand insertCommand2;
    // insert into responder Table
    SqlCommand insertCommand3;

    // update submission table
    insertCommand = new SqlCommand("UPDATE RFI_Submission_Table SET Status = 'Awaiting
Response' WHERE RFI_ID = (" + coorObj.RFI_ID + ")", con);
    insertCommand.ExecuteNonQuery();

    // get highest response_ID from table
    selectCommand = new SqlCommand("SELECT MAX(Response_ID) FROM RFI_Response_Table",
con);

    SqlDataReader d = selectCommand.ExecuteReader();
    d.Read();
    // to avoid confusion with RFI_ID the Response_ID will start at 100
    int response_ID = d[0] != DBNull.Value ? Convert.ToInt32(d[0]) + 1 : 101;
    d.Close();

    // update Response table with the response_ID
    insertCommand2 = new SqlCommand("INSERT INTO RFI_Response_Table(RFI_ID,
Response_ID) VALUES(@rf, @reid)", con);
    insertCommand2.Parameters.AddWithValue("@rf", coorObj.RFI_ID);
    insertCommand2.Parameters.AddWithValue("@reid", response_ID);
    insertCommand2.ExecuteNonQuery();

    foreach (string responder in responderListBox.SelectedItems)
    {
        Console.WriteLine("Foreach loop : {0}", responder);
        // add each responder to the Responder Table
        // the responses at this state will be 'awaiting answer'
        insertCommand3 = new SqlCommand("INSERT INTO Responder_Table(Response_ID,
Responder, ResponseApproved) VALUES(@reid, @resp, @ra)", con);
        insertCommand3.Parameters.AddWithValue("@reid", response_ID);
        insertCommand3.Parameters.AddWithValue("@ra", "awaiting answer");
    }
}

```

```

        insertCommand3.Parameters.AddWithValue("@resp", responder);
        insertCommand3.ExecuteNonQuery();
    }
    this.Close();
}

public void rejectClick(object sender, EventArgs e) {
    retrieveViewValues();
    if (coorObj.IsIncomplete) {
        MessageBox.Show("Form is incomplete", "User Error");
        return;
    }
    else {
        result = "rejected";

        SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);
        con.Open();
        // insert into Submission Table
        SqlCommand insertCommand;
        //get the highest RFI_ID

        insertCommand = new SqlCommand("UPDATE RFI_Submission_Table SET Comment = '' +
coorObj.Comments + '', Status = 'Awaiting Initiation' WHERE RFI_ID = (" + coorObj.RFI_ID +
")", con);
        insertCommand.ExecuteNonQuery();

        con.Close();
        this.Close();
    }
}

private void CoordinatorView_FormClosing(object sender, FormClosingEventArgs e)
{
    if (result == "cancelled")
    {
        DialogResult dr = MessageBox.Show("You are about to close without submitting.
Are you sure you want to close?", "Cancelling", MessageBoxButtons.YesNo);
        if(dr == DialogResult.Yes) {
            // just close
        }
        else {
            e.Cancel = true;
        }
    }
}
}

```

ObjectStructuresClass.cs file

```

using System;
using System.Data.SqlClient;
using System.Collections;

```

```

namespace RFIProcessWorkflowActivities
{
    class ResponseObject
    {

```

```

private string response;
private int response_ID;
private string responder;
private string responseApproved;
private string comments;
private Nullable<DateTime> responseDate;

public ResponseObject(
    string response,
    int response_ID,
    string responder,
    string responseApproved,
    string comments,
    DateTime responseDate
)
{
    this.Response = response;
    this.response_ID = response_ID;
    this.responder = responder;
    this.ResponseApproved = responseApproved;
    this.Comments = comments;
    this.responseDate = responseDate;
}

public ResponseObject(int response_ID, string responder)
{
    this.response_ID = response_ID;
    this.responder = responder;
    SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConne
ctionString);
    SqlCommand selectCommand = new SqlCommand("SELECT * FROM Responder_Table WHERE
Response_ID = (" + response_ID + ") AND Responder = '" + responder + "'", con);
    con.Open();
    SqlDataReader d = selectCommand.ExecuteReader();
    d.Read();
    this.ResponseApproved = d["ResponseApproved"].ToString();
    if (d["ResponseDate"] == DBNull.Value) {
        this.responseDate = DateTime.Now;
    }
    else {
        this.responseDate = Convert.ToDateTime(d["ResponseDate"].ToString());
    }

    this.Comments = d["Comments"].ToString();
    this.Response = d["Response"].ToString();
    d.Close();
    con.Close();
}

public bool IsIncomplete
{
    get {
        return
            String.IsNullOrEmpty(Response) ||
            !ResponseDate.HasValue ||
            String.IsNullOrEmpty(ResponseApproved) ||
            String.IsNullOrEmpty(Responder);
    }
}

```

```

    }

    public string Response
    {
        get { return response; }
        set { response=value; }
    }
    public int Response_ID {
        get { return response_ID; }
    }
    public string Responder {
        get { return responder; }
    }
    public string ResponseApproved {
        get { return responseApproved; }
        set { responseApproved = value; }
    }
    public string Comments {
        get { return comments; }
        set { comments = value; }
    }
    public Nullable<DateTime> ResponseDate {
        get { return responseDate; }
    }
}

/*
The InitiatorViewObject class contains all of the basic information
needed for an initiator view. The constructor retrieves information for
the associated database using the RFI_ID. This class is the parent class
(or base class) for the other view objects (Coordinator, Responder, etc.)
Its fields, and the methods used to access the database, can be reused by
its child class (or derived classes)
*/
class InitiatorViewObject
{
    // core fields that are necessary for RFI_ID
    private int rfi_id;
    private string project_id;
    private string projectTitle;
    private string status;
    private DateTime dateCreated;
    private Nullable<DateTime> requiredByDate = null;
    private string initiator;
    private string infoRequested;
    private string reason;

    // This constructor will either construct the object from the database
    // or - if the database is empty - it will create an empty object and
    // assign values to the required fields which the user cannot / should
    // not change

    // This constructor is reused by the child (derived) classes because
    // they share the inherited fields.
    public InitiatorViewObject(int rfi_id) {
        this.rfi_id = rfi_id;
        SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);

```



```

        SqlCommand selectCommand = new SqlCommand("SELECT * FROM RFI_Submission_Table
WHERE RFI_ID = (" + this.rfi_id + ")", con);
        con.Open();
        SqlDataReader d = selectCommand.ExecuteReader();
        d.Read();

        // If the RFI_ID that is passed in exists in the database, then the
        // info from the database is assigned to the object fields.
        if (d.HasRows) {
            status = d["Status"].ToString();
            dateCreated = Convert.ToDateTime(d["DateCreated"].ToString());
            initiator = d["Initiator"].ToString();
            Project_ID = d["Project_ID"].ToString();
            ProjectTitle = d["ProjectTitle"].ToString();
            InfoRequested = d["InfoRequested"].ToString();
            Reason = d["Reason"].ToString();
            RequiredByDate = Convert.ToDateTime(d["RequiredByDate"].ToString());
        }
        // If the RFI_ID does not exists in the database than the fields are
        // assigned default values.
        else {
            dateCreated = DateTime.Now;
            status = "Awaiting Initiation";
            initiator = "Initiator1";
        }
        d.Close();
        con.Close();
    }

    public virtual bool IsIncomplete {
        get {
            return
                String.IsNullOrEmpty(project_id) ||
                String.IsNullOrEmpty(projectTitle) ||
                String.IsNullOrEmpty(status) ||
                String.IsNullOrEmpty(infoRequested) ||
                String.IsNullOrEmpty(reason) ||
                !requiredByDate.HasValue;
        }
    }

    public int RFI_ID {
        get { return rfi_id; }
    }
    public string Project_ID {
        get { return project_id; }
        set { project_id = value; }
    }
    public string ProjectTitle
    {
        set { projectTitle = value; }
        get { return projectTitle; }
    }
    public string Status
    {
        get { return status; }
    }
    public DateTime DateCreated
    {

```

```

        get { return dateCreated; }
    }
    public Nullable<DateTime> RequiredByDate
    {
        get { return requiredByDate; }
        set { requiredByDate = value; }
    }
    public string Initiator
    {
        get { return initiator; }
    }
    public string InfoRequested
    {
        get { return infoRequested; }
        set { infoRequested = value; }
    }
    public string Reason
    {
        get { return reason; }
        set { reason = value; }
    }
}

/*
The CoordinatorViewObject class is used to contain all of the fields
that are relevant to the CoordinatorView. The class inherits from the
InitiatorViewObject to utilize existing code.
*/
class CoordinatorViewObject : InitiatorViewObject
{
    // The comments field needed in the Coordinator View
    private string comments;

    // The constructor uses the base constructor to read from the
    // database
    public CoordinatorViewObject(int rfi_id) : base(rfi_id) { }

    public override bool IsIncomplete {
        get {
            return base.IsIncomplete || String.IsNullOrEmpty(Comments);
        }
    }
    public string Comments
    {
        get { return comments; }
        set { comments = value; }
    }
}

/*
The ResponderViewObject class is used to contain all of the fields that
are relevant to the responder view. The class also inherits from the
InitiatorViewObject to utilize existing code
*/
class ResponderViewObject : InitiatorViewObject
{
    // fields unique to ResponderViewObject
    private ResponseObject responseObject;

```

```

// This constructor will call the base constructor to populate most of
// the fields. It will then read from the database and assign its unique
// fields values or, if the database is empty, populate them with default
// values
public ResponderViewObject(int rfi_id, string responder) : base(rfi_id)
{
    SqlConnection con = new
SqlConnection(RFIPProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);
    SqlCommand selectCommand = new SqlCommand("SELECT * FROM RFI_Response_Table WHERE
RFI_ID = (" + rfi_id + ")", con);
    con.Open();
    SqlDataReader d = selectCommand.ExecuteReader();
    d.Read();
    int response_ID = Convert.ToInt32(d["Response_ID"]);
    d.Close();
    con.Close();
    responseObject = new ResponseObject(response_ID, responder);
}

public override bool IsIncomplete {
    get {
        return base.IsIncomplete || responseObject.IsIncomplete;
    }
}

public ResponseObject ResponseObject
{
    get { return responseObject; }
    set { responseObject = value; }
}
}

/*
The ConsolidatorViewObject is used to contain all of the fields relevant
to the ConsolidatorView. It inherits directly from the ResponderViewObject
class, making it the child class (or grandchild class) of the
InitiatorViewObject.
*/
class ConsolidatorViewObject : InitiatorViewObject
{
    // fields unique to the consolidator view
    private ResponseObject responseObject;
    private string consolidatedResponse;
    Nullable<DateTime> approvalDate;

    public ConsolidatorViewObject(int rfi_id, string responder) : base(rfi_id)
    {
        SqlConnection con = new
SqlConnection(RFIPProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);
        SqlCommand selectCommand = new SqlCommand("SELECT * FROM RFI_Response_Table WHERE
RFI_ID = (" + rfi_id + ")", con);
        con.Open();
        SqlDataReader d = selectCommand.ExecuteReader();
        d.Read();
        int response_ID = Convert.ToInt32(d["Response_ID"]);
        if (d["ApprovalDate"] != DBNull.Value) {
            ApprovalDate = Convert.ToDateTime(d["ApprovalDate"].ToString());

```

```

    }
    else {
        ApprovalDate = DateTime.Now;
    }
    this.ConsolidatedResponse = d["ConsolidatedResponse"].ToString();
    d.Close();

    selectCommand = new SqlCommand("SELECT * FROM Responder_Table WHERE Response_ID =
(" + response_ID + ") AND Responder = '" + responder + "'", con);
    d = selectCommand.ExecuteReader();
    while (d.Read())
    {
        Console.WriteLine(" While loop responder: {0}", d["Responder"].ToString());
        responseObject = new ResponseObject(d["Response"].ToString(),
            Convert.ToInt32(d["Response_ID"]),
            d["Responder"].ToString(),
            d["ResponseApproved"].ToString(),
            d["Comments"].ToString(),
            Convert.ToDateTime(d["ResponseDate"].ToString())
        );
    }
    d.Close();
    con.Close();
}

public override bool IsIncomplete
{
    get
    {
        return
            base.IsIncomplete ||
            !ApprovalDate.HasValue;
    }
}

public ResponseObject ResponseObject
{
    get { return responseObject; }
    set { responseObject = value; }
}

public Nullable<DateTime> ApprovalDate
{
    get { return approvalDate; }
    set { approvalDate = value; }
}

public string ConsolidatedResponse
{
    get { return consolidatedResponse; }
    set { consolidatedResponse = value; }
}
}

class InitiatorResponseViewObject : InitiatorViewObject
{
    // The comments field needed in the Coordinator View
    private string comments;

```

```

// The constructor uses the base constructor to read from the
// database
public InitiatorResponseViewObject(int rfi_id) : base(rfi_id)
{
    SqlConnection con = new
SqlConnection(RFIProcessWorkflowActivities.Properties.Settings.Default.RFI_INFO_DATABASEConnec
tionString);
    SqlCommand selectCommand = new SqlCommand("SELECT * FROM RFI_Submission_Table
WHERE RFI_ID = (" + this.RFI_ID + ")", con);
    con.Open();
    SqlDataReader d = selectCommand.ExecuteReader();
    d.Read();

    // If the RFI_ID that is passed in exists in the database, then the
    // info from the database is assigned to the object fields.
    if (d.HasRows)
    {
        comments = d["Comment"].ToString();
    }
    // If the RFI_ID does not exists in the database than the fields are
    // assigned default values.
    else
    {
        comments = "Error. No comments found";
    }
    d.Close();
    con.Close();
}

public override bool IsIncomplete
{
    get
    {
        return base.IsIncomplete || String.IsNullOrEmpty(Comments);
    }
}
public string Comments
{
    get { return comments; }
}
}
}
}

```

Appendix C

Work Completed Under My Supervision to Support Validation of Conformance Checking (Tao Lue Wu, 2015)

1 Workflow

A workflow is comprised of a set of nodes, a set of directed edges, and two special nodes "Start" and "End". A node other than "Start" and "End" usually represents a task. A directed edge means that it is valid to begin one task after the other is finished. In a workflow, a valid sequence of tasks is a sequence of adjacent nodes that begins with the special node "Start" and ends with the special node "End".

1.1 Concept and Theory behind Workflow

Definition 1.1 (Workflow). A workflow is a directed graph $WF=(V, E, Start, End)$, where

- V is the set of all nodes in WF ,
- E is the set of all edges in WF .
- "Start" $\in V$, and
- "End" $\in V$.

Definition 1.2 (Path). Let $WF=(V, E, Start, End)$ be a workflow, a path in WF is a ordered sequence of nodes $(v_1, v_2, \dots, v_{n-1}, v_n)$, where $v_{i-1} \rightarrow v_i \in E$ for all $2 \leq i \leq n$.

Definition 1.3 (Reachability). Let $WF=(V, E, Start, End)$ be a workflow. For all $v, u \in V$, v can reach u in WF if there is a path from v to u . A node is defined to be always reachable from itself. We use $v.reachable_{WF}$ to denote all nodes that are reachable from node v in workflow WF .

Theorem 1.4 (Reachability and Simple Path). Let $WF=(V, E, Start, End)$ be a workflow. For all $v, u \in V$, v can reach u if and only if there exists a simple path from v to u .

Definition 1.5 (Test Path). A test path is path that starts with "Start" and ends with "End".

Definition 1.6 (Incident). In a workflow $WF=(V, E, Start, End)$. Let $e=v \rightarrow u \in E$. Then,

- e is incident with v and u
- e is incident from u
- v is incident to e

1.2 Alloy Implementation of Workflow

```
open util/graph[Step]

abstract sig Step {
  -- edges in the inheritant workflow/Contractor's workflow
  v : set Step,
  -- dominator tree
  idom2 : one Step,
  ipostdom2 : one Step
}

abstract sig Foundation extends Step {
  -- edges in the original workflow/IFP workflow
  w : set Foundation,
  -- dominator tree
  idom1 : one Step,
  ipostdom1 : one Step
}
-- distinguished Start and End nodes
one sig Start, End extends Foundation {}
```

In Alloy, we use *Step* to denote the set of all nodes that are involved in the model and use *v* to denote the set of all edges that are incident with nodes in *Step*. *Foundation* is a subset of *Step* and *w* is the set of all edges that are incident with nodes in *Foundation*. *w* and *v* are independent.

We will use *Foundation* and *w* to model a workflow. Notice that *Start* and *End* are predefined as nodes in *Foundation*. Therefore, a workflow can be modeled by $WF_1=(Foundation, w, Start, End)$.

1.3 Well-formed workflow

Definition 1.7 (Well-formed Workflow). **A well-formed workflow is a workflow** $WF=(V, E, Start, End)$

such that

- i) **No nodes in V can reach "Start" except "Start" itself, and**
 - ii) **"End" can not reach any nodes in V other than "End" itself,**
- and**
- iii) **For all nodes $v \in V$, "Start" can reach v , and**
 - iv) **For all nodes $v \in V$, v can reach End.**

Rule i) and ii) prevent incoming edges to "Start" or outgoing edges from "End". Rule iii) and iv) strictly forbid dangling nodes.

The following examples illustrate some workflows that are not well-formed.

Figure 1: Alloy Implementation of Workflow

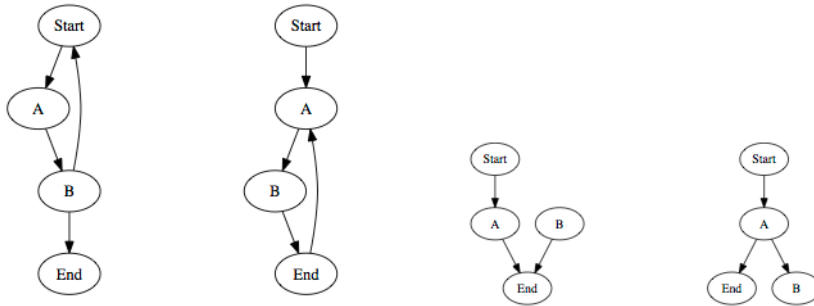


Figure 2: Non-well-formed workflows due to violations of rule i), ii), ii), iv) respectively, from left to right.

1.4 Alloy Implementation of Well-formed Workflow

```

pred wellFormed[nodes : set Step, e : Step->Step] {
  -- nodes includes Start and End
  Start in nodes
  End in nodes
  -- all nodes are reachable from Start
  nodes in Start.*e
  -- Start has no incoming edges
  no e.Start
  -- End is reachable from all nodes
  nodes in *e.End
  -- End has no outgoing edges
  no End.e
}

```

In the Alloy implementation, *nodes* denotes the set of all nodes (i.e. a set of *Step* in Alloy) and *e* denotes the set of all edges in the workflow. In this case, the edges are represented by a binary relation between two nodes (i.e. $e : Step \rightarrow Step$).

2 Dominator

The concept of dominator is borrowed from compiler ¹ ². Intuitively, the workflow dominators of a particular node v is the set of all nodes that appear in all paths from "Start" to v in the workflow. These nodes are unavoidable for v to be reached in the workflow and are thus referred the *dominators* of v , meaning the nodes that dominate v .

2.1 Concept and Theory behind Dominator

Definition 2.1 (Concept and Theory behind Dominator). **In a well-formed workflow $WF=(V, E, Start, End)$, node v dominates node u**

if and only if

- i) u is reachable from "Start", and**
- ii) u will no longer be reachable from "Start" if all edges that are incident with v are removed.**

If node v dominates node u in workflow WF , then v is a dominator of u in WF and u is an dominee of v in WF .

$v.dom_{WF}$ denotes all dominators of node v in workflow WF .

Definition 2.2 (Immediate Dominator). **In a well-formed workflow $WF=(V, E, Start, End)$, node v immediately dominates node u .**

if and only if

- i) v dominates u , and**
- ii) there exists no nodes $w \in V \setminus \{u, v\}$ such that v dominates w and w dominates u , and**
- iii) $v=u$ if and only if $v=Start$.**

If node v immediately dominates node u in workflow WF , then v is an immediate dominator of u in WF and u is an immediate dominee of v in WF .

$v.idom_{WF}$ denotes the set of all immediate dominators of v in workflow WF .

Corollary 2.3 (Self-Domination). **A node always dominates itself in a workflow.**

Lemma 2.4 (Transitivity of Domination). **In a well-formed workflow $WF=(V, E, Start, End)$, let $v, u, w \in V$. If v dominates u and u dominates w , then v dominates w .**

Lemma 2.5 (Inability of Mutual Domination).

In a well-formed workflow $WF=(V, E, Start, End)$, let $v, u \in V$ be

¹ C. N. Fischer, R. K. Cytron, and R. J. LeBlanc, Jr. *Crafting a Compiler*. Addison-Wesley, 2010; A. W. Appel and J. Palsberg. *Modern Compiler Implementation in Java*. Cambridge University Press, 2004; and M. L. Scott. *Programming Language Pragmatics*. Morgan Kaufmann, 3 edition, 2009

² L. Georgiadis, R. F. Werneck, R. E. Tarjan, S. Triantafyllis, and D. I. August. Finding dominators in practice. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA 2004)*, volume 3221 of *Lecture Notes in Computer Science*, pages 677–688. Springer-Verlag, 2004

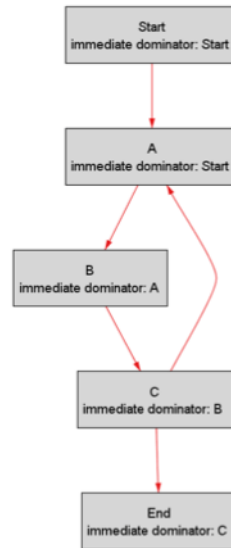


Figure 3: Example of immediate dominators of all nodes in a well-formed workflow

distinct nodes. Then v does not dominate u if u dominates v .

Proof. Suppose that there exists distinct nodes $v, u \in V$ such that v dominates u and u dominates v .

Since WF is well-formed, u is reachable from "Start" and so there exists a simple path p from "Start" to u .

v dominates u implies that p must pass through v . But then the subpath of p from "Start" to v is a path from "Start" to v without passing through u . A contradiction with u dominates v , which implies every path from "Start" to v must pass through u .

Therefore, v does not dominate u if u dominates v . □

Theorem 2.6 (Ordering of Dominators in Sequential workflow).

In a well-formed workflow $WF=(V,E,Start,End)$, let $v, u, w \in V$ be distinct nodes such that u and w are both dominators of v .

Then either w dominates u or u dominates w but not both.

Proof. Suppose that neither w dominates u nor u dominates w .

Then we can find a path p_1 from "Start" to v such that the first occurrence of w takes place before the first occurrence of u ; similarly, we can also find a path p_2 from "Start" to v such that the first occurrence of u takes place before the first occurrence of w .

Let the subpath of p_1 from "Start" to the first occurrence of w be p_3 .

We know that p_3 travels from "Start" to w without passing through u . Suppose that the last occurrence of u and w in p_2 is u , then there exists a path from u to v without passing through w ; combining it with the subpath of p_2 from "Start" to the first occurrence of u creates a path from "Start" to v without passing through w , a contradiction

since w dominates v . Thus, the last occurrence of u and w in p_2 must be w . Let the subpath of p_2 from the last occurrence of w to v be p_4 . But then combining p_3 and p_4 creates a path from "Start" to v without passing through u , a contradiction since u dominates v .

Therefore, either w dominates u or u dominates w . By **Lemma Inability of Mutual Domination**, we know that it is impossible that both are true.

Hence, either w dominates u or u dominates w but not both. \square

Corollary 2.7 (Uniqueness of Immediate Dominator in Sequential Workflow).

Let $WF=(V,E,Start,End)$ be a well-formed workflow.

Then for all $v \in V$, $|v.idom_{WF}| \leq 1$.

Proof. Suppose there exists distinct $u, w \in V$ such that $u, w \in v.idom_{WF}$.

This implies that u dominates v and w dominates v .

But then by **Theorem Ordering of Dominators in Sequential workflow**, either u dominates w or w dominates u , a contradiction since both u, w are immediate dominators of v .

Therefore, immediate dominator is unique. \square

Theorem 2.8 (Reflexive Transitive Closure of Immediate Dominator).

In a well-formed workflow $WF=(V,E,Start,End)$,

for all $v \in V$, $v.idom_{WF}^* = v.dom_{WF}$.

Proof.

$v.idom_{WF}^* \subseteq v.dom_{WF}$:

Let $u \in v.idom_{WF}^*$, then there exists a sequence of distinct nodes $v_1,$

$v_2, \dots, v_n \in V$ such that

u immediately dominates $v_1,$

v_1 immediately dominates $v_2,$

...

v_{n-1} immediately dominates $v_n,$

v_n immediately dominates v .

Since domination is transitive, this implies that u dominates v .

$v.dom_{WF} \subseteq v.idom_{WF}^*$:

Let $u \in v.dom_{WF}$. Suppose that $u \notin v.idom_{WF}^*$. Then u does not immediately dominate any nodes in $v.idom_{WF}^*$.

We want to find node $u' \in V$ such that

i) u' dominates v , and

ii) $u \in u'.idom_{WF}^*$, and

iii) u' does not immediately dominate any nodes in $v.idom_{WF}^*$.

To find such u' , we first find all immediate dominatees of u that dominates v , and then find all immediate dominatees of those immediate

dominatees that dominates v ; recursively, we can find u' as we know that u does not immediately dominates any $v' \in v.idom_{WF}^*$.

Since u' does not immediately dominate any nodes in $v.idom_{WF}^*$, u' does not immediately dominate v . So there exists some nodes $w \in V \setminus \{u', v\}$ such that u' dominates w and w dominates v .

Let W be the set of all such w . By Lemma Inability of Mutual Domination, there must be a node $w' \in W$ such that it is not dominated by any other nodes in W .

Since u' does not immediately dominate any nodes and thus not w' , there exists some $v' \in V \setminus \{u', w'\}$ such that u' dominates v' and v' dominates w' .

But then v' dominates v by Lemma Transitivity of Domination. So $v' \in W$ and $v' \neq w'$, a contradiction since w' is not dominated by any other nodes in W .

Therefore, $v.idom_{WF}^* \subseteq v.dom_{WF}$ by proof of contradiction. \square

2.2 Dominator Graph and Dominator Tree

Definition 2.9 (Dominator Graph). In a well-formed workflow $WF=(V_{WF},E_{WF},Start,End)$, the dominator graph of WF is a directed graph $DG=(V_{DG},E_{DG})$

where

- i) $V_{DG} = V_{WF}$, and
- ii) For all $v,u \in V_{DG}$, $v \rightarrow u \in E_{DG}$ if and only if v immediately dominates u in WF and $v \neq u$.

Corollary 2.10 (Reachability in Dominator Graph). Let $DG=(V_{DG},E_{DG})$ be a dominator graph of a well-formed workflow WF. Then for distinct $v, u \in V_{DG}$, $v \rightarrow u \in E_{DG}$ if and only if v dominates u .

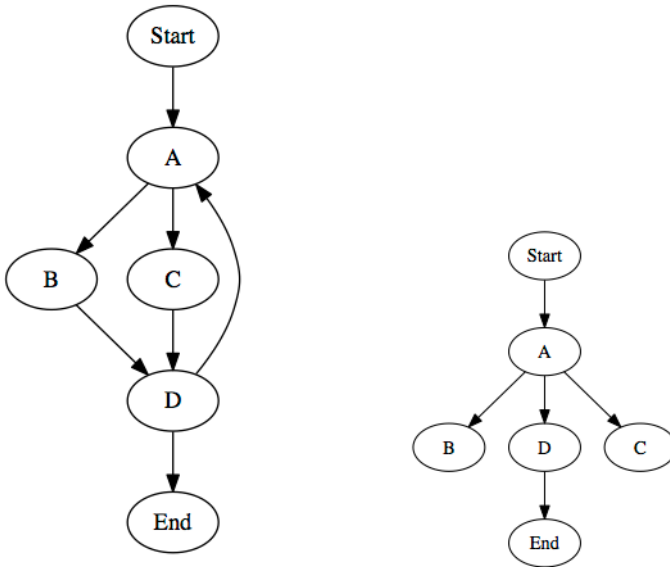


Figure 4: The right-hand side is the post-dominator graph of the workflow on the left-hand side.

Theorem 2.11 (Directed Acyclic Dominator Graph).

The dominator graph of a well-formed workflow is acyclic.

Proof. First, self-loop is forbidden by the definition of dominator graph.

Second, by Lemma Transitivity of Domination, forming a directed cycle between two distinct nodes in the dominator graph will result in those two nodes dominating each other. This is a contradiction with Lemma Inability of Mutual Domination. \square

Theorem 2.12 (Dominator Tree).

Let $WF=(V_{WF},E_{WF},Start,End)$ be a well-formed workflow,
then the dominator graph of WF is a tree.

Therefore, the dominator graph of a well-formed workflow is also called dominator tree.

Proof. A tree is graph with neither directed nor undirected cycles.

Let $DG=(V_{DG},E_{DG})$ be the dominator graph of a well-formed workflow WF. Suppose there is a cycle C in DG, then one of the following must be true:

i) C is a directed cycle. This is impossible since dominator graph is directed acyclic by **Theorem Acyclicity of Dominator Graph**.

ii) C is a undirected cycle. Then there exists distinct nodes $v, u, w \in V_{DG}$ such that $u \rightarrow v, w \rightarrow v \in E_{DG}$. This is also impossible since this implies that v has two immediate dominators u and w, which is in contradiction with **Lemma Uniqueness of Immediate Dominator in Sequential Workflow**.

Therefore, it is impossible to form neither a directed nor a undirected cycle.

Hence, the dominator graph of a well-formed workflow is a tree. \square

2.3 Alloy Implementation of Domination and Dominator Tree

```

-- compute dominator tree
-- e is all edges to consider
-- d is idom (immediate dominator) relation to be constrained
pred dominatorTree[n: set Step, e,d: Step->Step, begin,final: Step] {
  -- distinguished edge
  begin -> begin in d
  -- any node connected to Start is dominated by Start
  ~ (begin <: e) in d
  -- idom can at most be the inverse of e
  // no d - (~e + begin->begin)
  -- every node can get back to start following immediate dominators
  n in (^d).begin
  -- nothing dominates Start (except itself)
  begin.d =begin
  -- End dominates nothing
  no d.final
  -- nothing is the idom of itself except Start
  all x: (n-begin) | x != x.d
  -- x's immediate dominator is a dominator
  all x: n | let id=x.d | dominates[x, id, n, e, begin]
  -- x's immediate dominator is the closest dominator:
  -- there is no other node y that dominates x between x and x.d
  -- this property should apply to all nodes, including Start and End
  all x: n | let id=x.d | no y: n-id-x | dominates[x, y, n, e, begin] and dominates[y, id, n, e, begin]
}
pred dominatorTree[n: set Step, e,d: Step->Step] {
  dominatorTree[n, e, d, Start, End]
}

-- example to use pred dominatorTree
dominatorTree[Foundation, w, idom1]

-- true if x is dominated by y
-- y dominates x if all paths from start to x go through y
pred dominates[x, y: Step, n: set Step, e: Step->Step, begin: Step] {
  let e' = (e + (n <: iden) - (n->y + y->n)) | begin not in (^e').x
}
pred dominates[x, y: Step, n: set Step, e: Step->Step] {
  dominates[x, y, n, e, Start]
}

```

In the Alloy implementation, immediate domination is defined with predicate *dominate* (see the last line of the first *dominatorTree*, the one that takes 5 parameters). To use predicate *dominatorTree*, try *dominatorTree[Foundation, w, idom1]*, where *idom1* is the immediate dominator of workflow $WF_1=(Foundation, w, Start, End)$ (See the screenshot under section Alloy Implementation of Workflow).

3 Basic Conformance

Conformance relationship between workflows is used to defined workflow inheritance. The basic form of conformance is applied to well-formed workflows and can be illustrated with only the concept of dominator.

3.1 Concepts and Theory behind Basic Conformance

Definition 3.1 (Basic Conformance). Let $WF_1 = (V_1, E_1, Start_1, End_1)$ and $WF_2 = (V_2, E_2, Start_2, End_2)$ be well-formed workflows,

we say WF_2 is in basic conformance with WF_1 if and only if

- i) $V_1 \subseteq V_2$
- ii) For all $v \in V_1$, $v.dom_{WF_1} \subseteq v.dom_{WF_2}$.

Definition 3.2 (Basic Inherentance). If WF_2 is in conformance with WF_1 , then WF_2 is a basic inheritance of WF_1 .

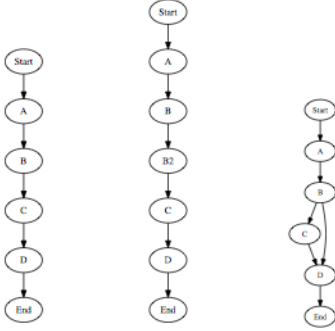


Figure 5: In the following figure, the middle workflow is in basic conformance with the left workflow. The right workflow is not in basic conformance with the left workflow since C is a dominator of D in the left workflow, but not in the right workflow.

Corollary 3.3 (Consistency of Dominator Tree Reachability in Basic Conformance).

Let WF_2 be a basic inherience of some well-formed workflow WF_1 .

Let $DG_1=(V_DG1, E_DG1)$ be the dominator tree of WF_1 and let

$DG_2=(V_DG2, E_DG2)$ be the dominator tree of WF_2 .

For all distinct nodes $v,u \in V_DG1$, v can reach u in DG_1 implies that v can reach u in $DG2$.

3.2 Alloy Implementation of Basic Conformance

In the Alloy Implementation, we make use of **Theorem Transitive Closure of Immediate Dominator** and find the set of all dominators of a node by computing the transitive closure of its immediate

dominator.

Furthermore, we do not need to check all dominators of every node in the original workflow. We can simplify this process by just making sure that the immediate dominator of every node in the original workflow continues to dominate that node in its basic inheritances.

Corollary 3.4 (Equivalence of Basic Conformance).

Let $WF_1 = (V_1, E_1, Start_1, End_1)$ and $WF_2 = (V_2, E_2, Start_2, End_2)$ be well-formed workflows.

Then WF_2 is in basic conformance with WF_1 if and only if

- i) $V_1 \subseteq V_2$
- ii) For all $v \in V_1$, $v.idom_{WF_1} \in v.dom_{WF_2}$.

-- new skip edges

```

fun skips[] : Step -> Step { { s,t : Step |

    s->t in v and
    t.idom1 not in (s + t.^(idom2))
/*
    or

    s->t in v and
    s.ipostdom1 not in (t + s.^(ipostdom2))
*/
}}

pred level2Conformance {
    // Level 2
    -- IFP workflow is well-formed
    wellFormed[Foundation, w]
    -- Contractor's derived workflow is well-formed
    wellFormed[Step, v]
    -- dominator tree of IFP workflow
    dominatorTree[Foundation, w, idom1]
    -- dominator tree of Contractor's derived workflow
    dominatorTree[Step, v, idom2]
}

```

Figure 6: Note that *Level 2 Conformance* is just another name for *Basic Conformance*.

Procedure 3.5 (Steps to Verify Basic Conformance).

Let $WF_1 = (V_1, E_1, Start_1, End_1)$ be the original workflow and $WF_2 = (V_2, E_2, Start_2, End_2)$ be a basic inheritance of WF_1 .

Step 1 Check whether both WF_1 and WF_2 are well-formed.

Step 2 Check whether $V_1 \subseteq V_2$.

Step 3 For all nodes $v \in V_1$, compute $v.idom_{WF_1}$ and $v.idom_{WF_2}$.

Step 3 For all nodes $v \in V_1$, check whether $v.idom_{WF_1} \in v.idom_{WF_2}^*$.

In summary, basic conformance ensures that if task A must be finished before task B in a workflow, then the same rule should be applied to all inheritances of that workflow.

4 Dominators are Not Enough

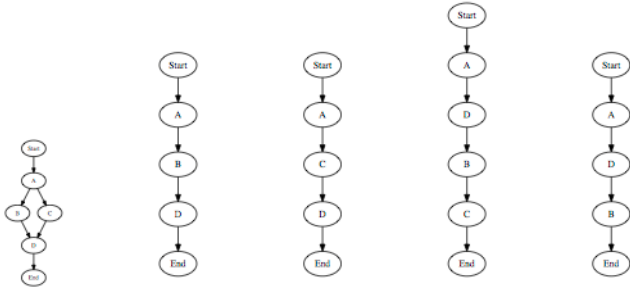


Figure 7: These workflows are named WF_1 , WF_2 , WF_3 , WF_4 , WF_5 , from left to right.

Let WF_1 be the first workflow from the left, WF_2 be the second workflow from the left, \dots , WF_5 be the fifth workflow from the left.

Let's make WF_1 the original workflow and the rest be its inheritances.

- WF_2 and WF_3) **Basic conformance** rejects WF_2 and WF_3 even though all test paths in WF_2 and WF_3 are valid test paths in WF_1 . We need a new type of conformance that allows node deletion and accepts WF_2 and WF_3 .
- WF_4) **Basic conformance** accepts WF_4 . However, node D comes after node B and C in original workflow WF_1 no matter which test path is taken. We want a second new type of conformance that enforces post-domination and rejects WF_4 .
- WF_5) **Basic conformance** rejects WF_5 as an inheritance of WF_1 because of node deletion. However, if we allow node deletion, we want a third new type of conformance that rejects WF_5 because of the same reason as WF_4 .

Before we introduce other types of conformances, we will first define the concept of post-dominator which will be used to define new conformance types that reject WF_4 and WF_5 .

5 Post-dominator

The intuition of post-dominator comes from reversing the concept of dominator. The only major difference is that post-dominator focuses on the nodes that are unavoidable to reach the "End" node.

5.1 Concepts and Theory behind Post-dominator

Definition 5.1 (Post-dominator). In a workflow $WF=(V, E, Start, End)$, node v post-dominates node u

if and only if

- u can reach "End", and
- u can no longer reach "End" if all edges that are incident with v are removed.

If node v post-dominates node u in workflow WF , then v is a post-dominator of u in WF and u is an post-dominatee of v in WF .

$v.postdom_{WF}$ denotes all post-dominators of node v in workflow WF .

Definition 5.2 (Post-dominator). In a workflow $WF=(V, E, Start, End)$, node v immediately post-dominates node u

if and only if

- i) v post-dominates u , and
- ii) there exists no nodes $w \in V \setminus \{u, v\}$ such that v post-dominates w and w post-dominates u , and
- iii) $v=u$ if and only if $v=End$.

If node v immediately post-dominates node u in workflow WF , then v is an immediate post-dominator of u and u is an immediate post-dominatee of v in WF .

$v.ipostdom_{WF}$ denotes the set of all immediate post-dominators of v in workflow WF .

The theorems, lemmas, and corollaries under the **Dominator section** will also hold for post-dominator. Proofs of them are omitted since they are very similar to the ones under the **Dominator section**.

Corollary 5.3 (Self-Post-domination). A node always post-dominates itself in a workflow.

Lemma 5.4 (Transitivity of Post-domination). In a well-formed workflow $WF=(V,E,Start,End)$, let $v, u, w \in V$. If v post-dominates u and u post-dominates w , then v post-dominates w .

Lemma 5.5 (Inability of Mutual Post-domination).

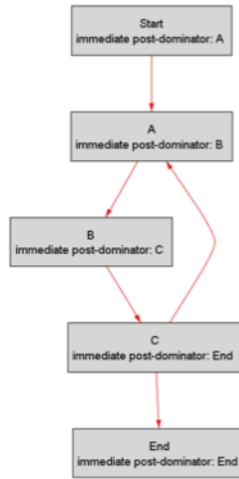


Figure 8: Example of immediate post-dominators of all nodes in a well-formed workflow

In a well-formed workflow $WF=(V,E,Start,End)$, let $v, u \in V$ be distinct nodes. Then v does not post-dominate u if u post-dominates v .

Theorem 5.6 (Ordering of Post-dominators in Sequential workflow).
In a well-formed workflow $WF=(V,E,Start,End)$, let $v, u, w \in V$ be distinct nodes such that u and w are both post-dominators of v . Then either w post-dominates u or u post-dominates w but not both.

Corollary 5.7 (Uniqueness of Immediate Post-dominator in Sequential Workflow).

Let $WF=(V,E,Start,End)$ be a well-formed workflow.

Then for all $v \in V$, $|v.ipostdom_{WF}| \leq 1$.

Theorem 5.8 (Reflexive Transitive Closure of Immediate Post-dominator).

In a well-formed workflow $WF=(V,E,Start,End)$,

for all $v \in V$, $v.ipostdom_{WF}^* = v.postdom_{WF}$.

Definition 5.9 (Post-dominator Graph). **In a well-formed workflow $WF=(V_{WF},E_{WF},Start,End)$, the post-dominator graph of WF is a directed graph $DG=(V_{DG},E_{DG})$**

where

i) $V_{DG} = V_{WF}$, and

ii) For all $v,u \in V_{DG}$, $v \rightarrow u \in E_{DG}$ if and only if v immediately post-dominates u in WF and $v \neq u$.

Corollary 5.10 (Reachability in Post-dominator Graph). **Let $DG=(V_{DG},E_{DG})$**

be a post-dominator graph of a well-formed workflow WF. Then for distinct $v, u \in V_{DGG}$, $v \rightarrow u \in E_{DGG}$ if and only if v post-dominates u .

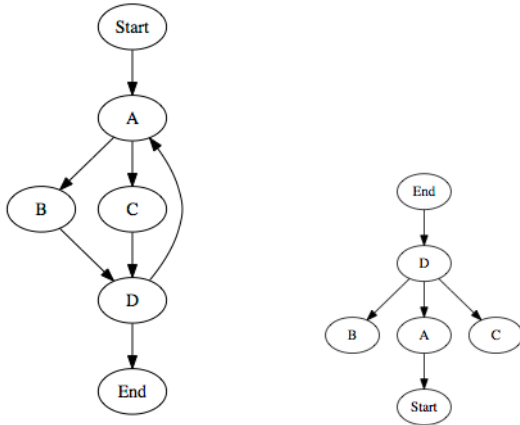


Figure 9: The right-hand side is the dominator graph of the workflow on the left-hand side.

Theorem 5.11 (Directed Acyclic Post-dominator Graph).
The post-dominator graph of a well-formed workflow is acyclic.

Theorem 5.12 (Post-dominator Tree).
Let $WF=(V,E,Start,End)$ be a well-formed workflow,
then the post-dominator graph of WF is a tree.
Therefore, the post-dominator graph of a well-formed workflow is
also called post-dominator tree.

5.2 Alloy Implementation of Post-Dominator Tree

```
-- post-dominator tree of IFP workflow
dominatorTree[Foundation, ~w, ipostdom1, End, Start]
```

In the Alloy implementation, we can use the same predicate that computes dominator tree to compute post-dominator tree. To do that, we simply reverse all edges in the original workflow and switch "Start" and "End".

6 Various Conformance Levels

To accept or reject different types of inheritances, we need to define different conformance levels. Conformance levels are ranked roughly based on their level of strictness.

6.1 Definitions of various Conformance Levels

Definition 6.1 (Level 1 Conformance). Let $WF_1 = (V_1, E_1, Start_1, End_1)$ and $WF_2 = (V_2, E_2, Start_2, End_2)$ be well-formed workflows,

we say WF_2 is in level 1 conformance with WF_1 if and only if

- i) For all $v \in V_1 \cap V_2$, $v.dom_{WF_1} \subseteq v.dom_{WF_2}$.

Definition 6.2 (Level 2 Conformance). See the definition of textbf-Basic Conformance.

Definition 6.3 (Level 3 Conformance). Let $WF_1 = (V_1, E_1, Start_1, End_1)$ and $WF_2 = (V_2, E_2, Start_2, End_2)$ be well-formed workflows,

we say WF_2 is in level 3 conformance with WF_1 if and only if

- i) For all $v \in V_1 \cap V_2$, $v.dom_{WF_1} \subseteq v.dom_{WF_2}$.
- ii) For all $v \in V_1 \cap V_2$, $v.postdom_{WF_1} \subseteq v.postdom_{WF_2}$.

Definition 6.4 (Level 4 Conformance). Let $WF_1 = (V_1, E_1, Start_1, End_1)$ and $WF_2 = (V_2, E_2, Start_2, End_2)$ be well-formed workflows,

we say WF_2 is in level 4 conformance with WF_1 if and only if

- i) $V_1 \subseteq V_2$
- ii) For all $v \in V_1$, $v.dom_{WF_1} \subseteq v.dom_{WF_2}$.
- iii) For all $v \in V_1$, $v.postdom_{WF_1} \subseteq v.postdom_{WF_2}$.

Now, recall the workflows examples in **Section Dominators are not enough**. Level 1 conformance is the least strict and accepts all of WF_2 , WF_3 , WF_4 , and WF_5 . Level 3 conformance rejects WF_5 because of inconsistency of post-domination instead of node deletion. Level 4 conformance is the most strict and rejects all of WF_2 , WF_3 , WF_4 , and WF_5 .

6.2 Alloy Implementation of Various Conformance Levels

```

-- new skip edges
fun skips[] : Step -> Step { { s,t : Step |

    s->t in v and
    t.idom1 not in (s + t.^(idom2))
/*
    or

    s->t in v and
    s.ipostdom1 not in (t + s.^(ipostdom2))
*/
}}

pred level2Conformance {
    // Level 2
    -- IFP workflow is well-formed
    wellFormed[Foundation, w]
    -- Contractor's derived workflow is well-formed
    wellFormed[Step, v]
    -- dominator tree of IFP workflow
    dominatorTree[Foundation, w, idom1]
    -- dominator tree of Contractor's derived workflow
    dominatorTree[Step, v, idom2]
}

-- new skip edges
fun skips[] : Step -> Step { { s,t : Step |

    s->t in v and
    t.idom1 not in (s + t.^(idom2))

    or

    s->t in v and
    s.ipostdom1 not in (t + s.^(ipostdom2))
}}

pred level3Conformance {
    // Level 3
    level1Conformance[]
    -- post-dominator tree of IFP workflow
    dominatorTree[Foundation, ~w, ipostdom1, End, Start]
    -- post-dominator tree of Contractor's derived workflow
    dominatorTree[Step.v+v.Step, ~v, ipostdom2, End, Start]
}

pred level4Conformance {
    // Level 4
    level2Conformance[]
    -- post-dominator tree of IFP workflow
    dominatorTree[Foundation, ~w, ipostdom1, End, Start]
    -- post-dominator tree of Contractor's derived workflow
    dominatorTree[Step, ~v, ipostdom2, End, Start]
}

```

Function *skips* detects inconformance in the workflow and marks these edges as "skips". The post-dominatation part of the *skips* function should be commented out when using *levelConformance* or *level2Conformance*.

Appendix D

Alloy Code for RFI Workflow Conformance Checking

```
open util/graph[Step]

abstract sig Step {
  -- edges in the Contractor's workflow
  v : set Step,
  -- dominator tree
  idom2 : one Step,
  ipostdom2 : one Step
}

abstract sig Foundation extends Step {
  -- edges in the IFP workflow
  w : set Foundation,
  -- dominator tree
  idom1 : one Step,
  ipostdom1 : one Step
}

-- distinguished Start and End nodes
one sig Start, End extends Foundation {}

abstract sig Concrete extends Step {}

-- all edges
fun edges[] : Step -> Step { w + v }

fact ConformanceLevel {
  // level1Conformance[]
  // level2Conformance[]
  // level3Conformance[]
  level4Conformance[]
}

-- compute dominator tree
-- e is all edges to consider
-- d is idom (immediate dominator) relation to be constrained
pred dominatorTree[n: set Step, e,d : Step->Step, begin,final : Step] {
  -- distinguished edge
  begin -> begin in d
  -- any node connected to Start is dominated by Start
  ~(begin <: e) in d
  -- idom can at most be the inverse of e
  // no d - (~e + begin->begin)
  -- every node can get back to start following immediate dominators
  n in (^d).begin
  -- nothing dominates Start (except itself)
  begin.d =begin
  -- End dominates nothing
  no d.final
  -- nothing is the idom of itself except Start
  all x : (n-begin) | x != x.d
  -- x's immediate dominator is a dominator
  all x : n | let id=x.d | dominates[x, id, n, e, begin]
  -- x's immediate dominator is the closest dominator:
  -- there is no other node y that dominates x between x and x.d
```

```

    -- this property should apply to all nodes, including Start and End
    all x : n | let id=x.d | no y : n-id-x | dominates[x, y, n, e, begin] and dominates[y, id,
n, e, begin]
}
pred dominatorTree[n: set Step, e,d : Step->Step] {
    dominatorTree[n, e, d, Start, End]
}

-- true if x is dominated by y
-- y dominates x if all paths from start to x go through y
pred dominates[x, y : Step, n : set Step, e : Step->Step, begin: Step] {
    let e' = (e + (n <: iden) - (n->y + y->n)) | begin not in (^e').x
}
pred dominates[x, y : Step, n : set Step, e : Step->Step] {
    dominates[x, y, n, e, Start]
}

-- dominates_alt should does the same check as dominates
-- dominates_alt is used in assert DomIsReinforced and IDomIsReinforced
pred dominates_alt[x, y : Step, n : set Step, e : Step->Step, begin: Step] {
    let e' = (e + (n <: iden) - ((y <: e) + (e :> y) + (y->y)) ) | not reachable[x, begin,e']
}
pred dominates_alt[x, y : Step, n : set Step, e : Step->Step] {
    dominates_alt[x, y, n, e, Start]
}

-- x is reachable from y
pred reachable[x, y : Step, e : Step->Step]{
    x in y.(^e)
}

pred wellFormed[nodes : set Step, e : Step->Step] {
    -- nodes includes Start and End
    Start in nodes
    End in nodes
    -- all nodes are reachable from Start
    nodes in Start.*e
    -- Start has no incoming edges
    no e.Start
    -- End is reachable from all nodes
    nodes in *e.End
    -- End has no outgoing edges
    no End.e
}

pred level1Conformance {
    // Level 1
    -- IFP workflow is well-formed
    wellFormed[Foundation, w]
    -- Contractor's derived workflow is well-formed
    wellFormed[Step.v+v.Step, v]
    -- dominator tree of IFP workflow
    dominatorTree[Foundation, w, idom1]
    -- dominator tree of Contractor's derived workflow
    dominatorTree[Step.v+v.Step, v, idom2]
}

pred level2Conformance {
    // Level 2

```



```

-- IFP workflow is well-formed
wellFormed[Foundation, w]
-- Contractor's derived workflow is well-formed
wellFormed[Step, v]
-- dominator tree of IFP workflow
dominatorTree[Foundation, w, idom1]
-- dominator tree of Contractor's derived workflow
dominatorTree[Step, v, idom2]
}

pred level3Conformance {
  // Level 3
  level1Conformance[]
  -- post-dominator tree of IFP workflow
  dominatorTree[Foundation, ~w, ipostdom1, End, Start]
  -- post-dominator tree of Contractor's derived workflow
  dominatorTree[Step.v+v.Step, ~v, ipostdom2, End, Start]
}

pred level4Conformance {
  // Level 4
  level2Conformance[]
  -- post-dominator tree of IFP workflow
  dominatorTree[Foundation, ~w, ipostdom1, End, Start]
  -- post-dominator tree of Contractor's derived workflow
  dominatorTree[Step, ~v, ipostdom2, End, Start]
}

one sig Response_Close_Out, Consolidate_and_Endorse, Verify_Details, Respond,
Sufficient_Details, Initial_RFI, Clarify, Approve extends Foundation {}

one sig Respond_Directly, Select_Coordinator, Select_Participants,
Clarification_Required_Responders, Verify_for_Clarification,
Response_Notification_To_Stakeholders, Verify_and_Update_Participants,
Clarification_Required_Consolidator extends Concrete {}

fact W1defn {
  w = {Start -> Initial_RFI + Initial_RFI -> Verify_Details + Verify_Details ->
Sufficient_Details + Sufficient_Details -> Respond + Sufficient_Details -> Clarify + Respond -
> Consolidate_and_Endorse + Consolidate_and_Endorse -> Approve + Approve -> Response_Close_Out
+ Response_Close_Out -> End + Clarify -> Verify_Details}
}

fact W2defn {
  v = {Start -> Initial_RFI + Initial_RFI -> Select_Coordinator + Select_Coordinator ->
Verify_Details + Verify_Details -> Sufficient_Details + Sufficient_Details -> Respond_Directly
+ Respond_Directly -> Response_Close_Out + Respond_Directly -> Select_Participants +
Response_Close_Out -> Response_Notification_To_Stakeholders + Select_Participants ->
Verify_and_Update_Participants + Response_Notification_To_Stakeholders -> End +
Verify_and_Update_Participants -> Respond + Respond -> Clarification_Required_Responders +
Clarification_Required_Responders -> Verify_for_Clarification + Verify_for_Clarification ->
Clarification_Required_Consolidator + Clarification_Required_Consolidator -> Clarify +
Clarification_Required_Consolidator -> Consolidate_and_Endorse + Consolidate_and_Endorse ->
Approve + Sufficient_Details -> Clarify + Clarification_Required_Responders ->
Consolidate_and_Endorse + Clarify -> Verify_Details + Approve -> Respond + Approve ->
Response_Close_Out}
}

// open workflow

```

```

-- preserved edges
fun preserved[] : Step -> Step { w & v }

-- deleted edges
fun deleted[] : Step -> Step { w - v }

-- new legal forward edges
fun forward[] : Step -> Step { v - w - backw - backv - skips }

-- new skip edges
fun skips[] : Step -> Step { { s,t : Step |
  -- source -> target is an edge in the customized workflow and
  s->t in v and (
    -- target's original immediate dominator is not in its new dominators
    t.idom1 not in (s + t.^idom2)
    or
    -- or source's original immediate post-dominator is not in its new post-dominators
    s.ipostdom1 not in (t + s.^ipostdom2)
  ) } }

-- new back edges
fun backv[] : Step -> Step { { s,t : Step |
  -- it's a new edge
  s->t in (v-w) and
  -- target is a dominator of source
  t in s.^(idom1+idom2)} }

-- back edges in Foundation (might be deleted)
fun backw[] : Step -> Step { { s,t : Step |
  -- it's an old edge
  s->t in w and
  -- target is a dominator of source
  t in s.^idom1 }}

run {}

```

Appendix E

Translator.java Documentation

Main

public static void main(**String[]** args)

The main method takes the file names of two xaml files (the original workflow and the derived workflow). It then calls the translate method to get the edges of the two workflows in alloy format. At last, it prints a complete als file to System.out (Standard Output of Translator.java) and the corresponding thm file to System.err (Standard Error of Translator.java).

note:

- als file is the file format used by alloy.
- A thm file specifies the format of graphical representation used by an als file.
- xaml is a special way to represent a workflow with xml format in Microsoft Windows Workflow Foundation. Therefore, an xaml file can be parsed by an xml parser. In the translate method, xml parser XPath is used to parse the xaml file.
(<http://docs.oracle.com/javase/8/docs/api/javax/xml/xpath/package-summary.html>,
<http://www.w3schools.com/xsl/default.asp>)

Parameters:

- args[0] - the path to the xaml file that represents the original workflow
- args[1] - the path to the xaml file that represents the derived workflow

Returns:

- void

Side Effect:

- Prints a complete als file to System.out (Standard Output of Translator.java)
- Prints the corresponding thm file of the als file to System.err (Standard Error of Translator.java)

Translate

String translate(Document doc, HashSet<String> foundations, HashSet<String> concretes)

throws Exception

This method translates a workflow from a xaml file to edges in alloy format.

Whenever a new node is encountered in the current workflow, this method will check whether the node exists in foundations. If not, the new node will be added to concretes.

This method automatically identifies whether the workflow in the xaml file is in FlowChart format or StateMachine format, and then calls translateFlowChart or translateStateMachine corresponding.

note:

- FlowChart and StateMachine are two different formats used by Microsoft Windows Workflow Foundation to represent a workflow in xaml. ([https://msdn.microsoft.com/en-us/library/dd489437\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd489437(v=vs.110).aspx))

- xaml is a special way to represent a workflow with xml format in Microsoft Windows Workflow Foundation. Therefore, an xaml file can be parsed by an xml parser. In the translate method, xml parser XPath is used to parse the xaml file.
(<http://docs.oracle.com/javase/8/docs/api/javafx/xml/xpath/package-summary.html>,
<http://www.w3schools.com/xsl/default.asp>)

Parameters:

- doc - the Document object that is used to represent the source xaml file
- foundations - set of nodes that appears in the original workflow, use foundations = null if you are translating the original workflow.
- concretes - set of nodes that appears in the current workflows but are not contained by foundations.

Returns:

- A String that represents all edges of the current workflow in alloy format

translateFlowChart

String translateFlowChart(Document doc, HashSet<String> foundations, HashSet<String> concretes)

throws Exception

This method translates a FlowChart workflow from an xaml file (parameter doc) to edges in alloy format.

Warning: The workflow within the xaml file (parameter doc) **must be in FlowChart format**. Behavior is undefined if this method is called against an xaml file that contains other types of workflow.

Whenever a new node is encountered in the current workflow, this method will check whether the node exists in foundations. If not, the new node will be added to concretes.

note:

- FlowChart and StateMachine are two different formats used by Microsoft Windows Workflow Foundation to represent a workflow in xaml. ([https://msdn.microsoft.com/en-us/library/dd489437\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd489437(v=vs.110).aspx))
- xaml is a special way to represent a workflow with xml format in Microsoft Windows Workflow Foundation. Therefore, an xaml file can be parsed by an xml parser. In the translate method, xml parser XPath is used to parse the xaml file.
(<http://docs.oracle.com/javase/8/docs/api/javafx/xml/xpath/package-summary.html>,
<http://www.w3schools.com/xsl/default.asp>)

Parameters:

- doc - the Document object that is used to represent the source xaml file
- foundations - set of nodes that appears in the original workflow, use foundations = null if you are translating the original workflow.
- concretes - set of nodes that appears in the current workflows but are not contained by foundations.

Returns:

- A String that represents all edges of the current workflow in alloy format

translateStateMachine

String translateStateMachine(Document doc, HashSet<String> foundations, HashSet<String> concretes)
throws Exception

This method translates a StateMachine workflow from an xaml file (parameter doc) to edges in alloy format.

Warning: The workflow within the xaml file (parameter doc) **must be in StateMachine format**. Behavior is undefined if this method is called against an xaml file that contains other types of workflow.

Whenever a new node is encountered in the current workflow, this method will check whether the node exists in foundations. If not, the new node will be added to concretes.

note:

- FlowChart and StateMachine are two different formats used by Microsoft Windows Workflow Foundation to represent a workflow in xaml. ([https://msdn.microsoft.com/en-us/library/dd489437\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd489437(v=vs.110).aspx))
- xaml is a special way to represent a workflow with xml format in Microsoft Windows Workflow Foundation. Therefore, an xaml file can be parsed by an xml parser. In the translate method, xml parser XPath is used to parse the xaml file. (<http://docs.oracle.com/javase/8/docs/api/javafx/xml/xpath/package-summary.html>, <http://www.w3schools.com/xsl/default.asp>)

Parameters:

- doc - the Document object that is used to represent the source xaml file
- foundations - set of nodes that appears in the original workflow, use foundations = null if you are translating the original workflow.
- concretes - set of nodes that appears in the current workflows but are not contained by foundations.

Returns:

- A String that represents all edges of the current workflow in alloy format

produceThm

void produceThm(final PrintWriter pw, Collection<String> nodes)

This method writes a thm file to PrintWriter pw for a workflow with nodes in Collection nodes.

Parameters:

- PrintWriter pw - the output source of the theme file
- Collection<String> nodes - all nodes presented in the workflow

Return:

- void

Side Effect:

- writes a thm file to PrintWriter pw for a workflow with nodes in Collection nodes.

Appendix F

Translator.java Code

```
import javax.xml.parsers.*;
import javax.xml.xpath.*;

import org.w3c.dom.*;
import org.xml.sax.InputSource;

import java.lang.StringBuffer;
import java.util.AbstractMap;
import java.util.Queue;
import java.util.LinkedList;
import java.util.HashSet;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.Collection;
import java.util.LinkedList;
import java.util.AbstractMap.SimpleEntry;
import java.io.PrintStream;
import java.io.StringReader;
import java.io.File;
import java.io.PrintWriter;
import java.io.FileNotFoundException;

public class Translator {

    public static final String SAP_IDREF = "sap2010:WorkflowViewState.IdRef";

    // public static void produceThm(String thmFileName, Collection<String>
    // nodes){
    public static void produceThm(final PrintWriter pw, Collection<String> nodes) {
        pw.println("<?xml version='1.0'?>");
        pw.println("<alloy>\n");

        pw.println("<view nodetheme='Martha' hidePrivate='no'>\n");
        pw.println("<defaultnode/>\n");
        pw.println("<defaultedge/>\n");

        pw.println("<node>");
        pw.println("\t<type name='Int'>");
        pw.println("\t<type name='String'>");
        pw.println("\t<type name='Univ'>");
        pw.println("\t<type name='univ'>");
        pw.println("\t<type name='seq/Int'>");
        pw.println("</node>\n");

        pw.println("<node label='Step'>");
        pw.println("\t<type name='Step'>");
        pw.println("</node>\n");

        pw.println("<node color='Gray' label='Foundation'>");
        pw.println("\t<type name='Foundation'>");
        pw.println("</node>\n");
    }
}
```

```

pw.println("<node color=\"White\" label=\"Concrete\">");
pw.println("\t<type name=\"Concrete\"/>");
pw.println("</node>\n");

pw.println("<node label=\"Start\">");
pw.println("\t<type name=\"Start\"/>");
pw.println("</node>\n");

pw.println("<node label=\"End\">");
pw.println("\t<type name=\"End\"/>");
pw.println("</node>\n");
for (String node : nodes) {
    pw.println("<node label=\"" + node + "\">");
    pw.println("\t<type name=\"" + node + "\"/>");
    pw.println("</node>\n");
}
pw.println("<edge color=\"Black\" label=\"\">");
pw.println("\t<relation name=\"$preserved\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("</edge>\n");

pw.println("<edge color=\"Blue\" label=\"\">");
pw.println("\t<relation name=\"$back\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("</edge>\n");

pw.println("<edge color=\"Gray\" label=\"\">");
pw.println("\t<relation name=\"$deleted\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("</edge>\n");

pw.println("<edge color=\"Green\" label=\"\">");
pw.println("\t<relation name=\"$forward\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("</edge>\n");

pw.println("<edge color=\"Red\" label=\"skips\">");
pw.println("\t<relation name=\"$skips\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("</edge>\n");

pw.println("<edge visible=\"no\">");
pw.println("\t<relation name=\"$backw\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("\t<relation name=\"$edges\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("\t<relation name=\"$v\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("\t<relation name=\"$w\"> <type name=\"Foundation\"/> <type name=\"Foundation\"/>
</relation>");
pw.println("</edge>\n");

pw.println("<edge visible=\"no\" attribute=\"yes\">");
pw.println("\t<relation name=\"idom1\"> <type name=\"Foundation\"/> <type name=\"Step\"/> </relation>");
pw.println("\t<relation name=\"idom2\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("\t<relation name=\"ipostdom1\"> <type name=\"Foundation\"/> <type name=\"Step\"/>
</relation>");
pw.println("\t<relation name=\"ipostdom2\"> <type name=\"Step\"/> <type name=\"Step\"/> </relation>");
pw.println("</edge>\n");

pw.println("</view>\n");

pw.println("</alloy>\n");

```

```

        pw.close();
    }

    public static Node getFirstChild(Node node) throws Exception {
        if (node == null) {
            return null;
        }

        XPath xpath = XPathFactory.newInstance().newXPath();
        NodeList nodeList = (NodeList) xpath.compile("./*").evaluate(node,
            XPathConstants.NODESET);
        Node firstNode = null;
        if (nodeList != null && nodeList.getLength() > 0) {
            firstNode = nodeList.item(0);
        }
        return firstNode;
    }

    public static String getDisplayName(Element e) throws Exception {
        String displayName;
        if (e.getTagName().equals("FlowStep")) {
            e = (Element) getFirstChild((Node) e);
        }
        if (e.hasAttribute("DisplayName"))
            displayName = e.getAttribute("DisplayName");
        else {
            displayName = e.getAttribute(Translator.SAP_IDREF);
        }
        return displayName.replace(' ', '_');
    }

    public static void addEdge(String currStep, Element nextStepElement,
        HashSet<String> foundations, HashSet<String> concretes,
        HashMap<String, String> XNameToDisplayName, StringBuffer strBuffer)
        throws Exception {

        String nextStep = getDisplayName(nextStepElement);

        XNameToDisplayName
            .put(nextStepElement.getAttribute("x:Name"), nextStep);
        if (foundations == null || !foundations.contains(nextStep)) {
            concretes.add(nextStep);
        }
        strBuffer.append(currStep);
        strBuffer.append(" -> ");
        strBuffer.append(nextStep);
        strBuffer.append(" + ");
    }

    public static void addEdge(String currStep, String nextStep,
        StringBuffer strBuffer) {
        strBuffer.append(currStep);
        strBuffer.append(" -> ");
        strBuffer.append(nextStep);
        strBuffer.append(" + ");
    }

```



```

}

public static String getAlloyEdges(Document doc, HashSet<String> foundations,
    HashSet<String> concretes) throws Exception {

    XPath xpath = XPathFactory.newInstance().newXPath();

    Node obj = (Node) xpath.compile("/Activity/Flowchart").evaluate(doc,
        XPathConstants.NODE);
    if (obj != null) {
        return FlowChartToAlloyEdges(doc, foundations, concretes);
    }
    obj = (Node) xpath.compile("/Activity/StateMachine").evaluate(doc,
        XPathConstants.NODE);
    if (obj != null) {
        return StateMachineToAlloyEdges(doc, foundations, concretes);
    }
    String docURI = (doc.getDocumentURI() == null) ? (" " : (" "
        + doc.getDocumentURI() + " ");
    throw new Exception("file" + docURI
        + "contains neither a Flowchart nor StateMachine");
}

public static String FlowChartToAlloyEdges(Document doc,
    HashSet<String> foundations, HashSet<String> concretes)
    throws Exception {

    StringBuffer outputStrBuffer = new StringBuffer();

    NodeList neighbours;
    Node startNode;
    Element startElement;
    Node node;

    XPath xpath = XPathFactory.newInstance().newXPath();
    XPathExpression xexpr = xpath
        .compile("/Activity/Flowchart/Flowchart.StartNode");
    startNode = (Node) xexpr.evaluate(doc, XPathConstants.NODE);

    Queue<Node> Q = new LinkedList<Node>();
    HashMap<String, String> XNameToDisplayName = new HashMap<String, String>();
    LinkedList<AbstractMap.SimpleEntry<String, String>> delayedEdges = new
LinkedList<AbstractMap.SimpleEntry<String, String>>();

    String displayName;
    String currStep = "Start", nextStep;

    if (startNode != null) {
        // StartNode found as tag /Activity/Flowchart/Flowchart.StartNode
        neighbours = (NodeList) xpath.compile("./FlowDecision|./FlowStep")
            .evaluate(startNode, XPathConstants.NODESET);
        if (neighbours != null && neighbours.getLength() > 0) {
            for (int i = 0; i < neighbours.getLength(); i++) {
                Q.add(neighbours.item(i));
                addEdge(currStep, (Element) neighbours.item(i),
                    foundations, concretes, XNameToDisplayName,

```

```

        outputStrBuffer);
    }
} else {
    addEdge(currStep, "End", outputStrBuffer);
}
} else {
    // StartNode not found as a tag
    // get StartNode as an attribute under tag /Activity/Flowchart
    neighbours = (NodeList) xpath.compile(
        "/Activity/Flowchart/@StartNode").evaluate(startNode,
        XPathConstants.NODESET);
    if (neighbours != null && neighbours.getLength() > 0) {
        for (int i = 0; i < neighbours.getLength(); i++) {
            String[] strArray = neighbours.item(i).getTextContent().split(" ");
            String xName = strArray[1].substring(0,
                strArray[1].length() - 1);
            delayedEdges.add(new AbstractMap.SimpleEntry<String, String>(currStep,
                xName));
        }
        /*for (int i = 0; i < neighbours.getLength(); i++) {
            delayedEdges.add(new AbstractMap.SimpleEntry<String, String>(currStep,
                neighbours.item(i).getTextContent()));
        }*/
        neighbours = (NodeList) xpath.compile(
            "/Activity/Flowchart/FlowDecision|/Activity/Flowchart/FlowStep")
            .evaluate(startNode, XPathConstants.NODESET);
        for (int i = 0; i < neighbours.getLength(); i++) {
            Q.add(neighbours.item(i));
        }
    } else {
        addEdge(currStep, "End", outputStrBuffer);
    }
}

while (!Q.isEmpty()) {
    startNode = Q.remove();
    startElement = (Element) startNode;
    currStep = getDisplayName(startElement);

    // get edges from tags
    neighbours = (NodeList) xpath.compile(
        "./FlowDecision.False/*|./FlowDecision.True/*|./FlowStep.Next/*").
        evaluate(startNode, XPathConstants.NODESET);
    boolean hasEdges = false;
    if (neighbours != null && neighbours.getLength() > 0) {
        for (int i = 0; i < neighbours.getLength(); i++) {
            Element neighbourElement = (Element) neighbours.item(i);
            if (neighbourElement.getTagName() != null) {
                if (neighbourElement.getTagName().equals("FlowDecision")
                    || neighbourElement.getTagName().equals("FlowStep")) {
                    hasEdges = true;
                    Q.add(neighbours.item(i));
                    addEdge(currStep, neighbourElement, foundations,

```

```

        concretes, XNameToDisplayName,
        outputStrBuffer);

    } else if (neighbourElement.getTagName().equals("x:Reference")) {

        hasEdges = true;
        delayedEdges.add(new AbstractMap.SimpleEntry<String, String>(
            currStep,
            neighbours.item(i).getTextContent()));

    }

}

// get edges from attributes
neighbours = null;
if (startElement.getTagName() != null) {
    if (startElement.getTagName().equals("FlowDecision")) {

        neighbours = (NodeList) xpath.compile("./@True|./@False")
            .evaluate(startNode, XPathConstants.NODESET);

    } else if (startElement.getTagName().equals("FlowStep")) {

        neighbours = (NodeList) xpath.compile("./@Next")
            .evaluate(startNode, XPathConstants.NODESET);

    }

}
if (neighbours != null && neighbours.getLength() > 0) {
    hasEdges = true;
    for (int i = 0; i < neighbours.getLength(); i++) {
        String[] strArray = neighbours.item(i).getTextContent().split(" ");
        String xName = strArray[1].substring(0,
            strArray[1].length() - 1);
        delayedEdges.add(new AbstractMap.SimpleEntry<String, String>(currStep,
            xName));
    }
}
if (!hasEdges) {
    addEdge(currStep, "End", outputStrBuffer);
}

// print back/cross edges that has not yet been discovered when
// first seen
for (AbstractMap.SimpleEntry<String, String> edge : delayedEdges) {
    outputStrBuffer.append(edge.getKey());
    outputStrBuffer.append(" -> ");
    outputStrBuffer.append(XNameToDisplayName.get(edge.getValue()));
    outputStrBuffer.append(" + ");
}

// remove " + " at the end of the StringBuffer
if (outputStrBuffer.length() >= 3

```

```

        && outputStrBuffer.charAt(outputStrBuffer.length() - 1) == ' '
        && outputStrBuffer.charAt(outputStrBuffer.length() - 2) == '+'
        && outputStrBuffer.charAt(outputStrBuffer.length() - 3) == ' ') {

            outputStrBuffer.setLength(outputStrBuffer.length() - 3);
        }

    return outputStrBuffer.toString();
}

public static String StateMachineToAlloyEdges(Document doc,
        HashSet<String> foundations, HashSet<String> concretes)
        throws Exception {

    StringBuffer outputStrBuffer = new StringBuffer();

    Node initialState;
    NodeList initialNodes;

    Node node;
    Element e;
    NodeList transitions;

    XPath xpath = XPathFactory.newInstance().newXPath();
    XPathExpression xexpr = xpath
        .compile("/Activity/StateMachine/StateMachine.InitialState/State");
    initialState = (Node) xexpr.evaluate(doc, XPathConstants.NODE);

    Queue<Node> Q = new LinkedList<Node>();
    HashMap<String, String> XNameToDisplayName = new HashMap<String, String>();
    HashMap<String, String> delayedEdges = new HashMap<String, String>();

    String displayName;
    String currState = "Start", nextState;
    // get edges from the "Start" node
    // i.e. the initial state
    if (initialState == null) {
        // initialState is not defined yet
        // we can only get the x:reference of the initial state
        xexpr = xpath.compile("/Activity/StateMachine");
        node = (Node) xexpr.evaluate(doc, XPathConstants.NODE);
        e = (Element) node;
        String attr = e.getAttribute("InitialState");
        if (attr != null && !attr.isEmpty() && attr.trim().length() > 0) {
            String[] strArray = attr.split(" ");
            String xName = strArray[1].substring(0,
                strArray[1].length() - 1);

            // save the x:reference of the initial state to delayedEdges
            // which will be added to the output StringBuffer at the end
            delayedEdges.put(xName, currState);
        }

        // get states that are defined in
        initialNodes = (NodeList) xpath.compile("./State").evaluate(
            node, XPathConstants.NODESET);
    }
}

```

```

for (int i = 0; i < initialNodes.getLength(); i++) {
    node = initialNodes.item(i);
    e = (Element) node;

    Q.add(node);
    displayName = e.getAttribute("DisplayName").replace(' ',
        '_');
    XNameToDisplayName.put(e.getAttribute("x:Name"),
        displayName);
    if (foundations == null
        || !foundations.contains(displayName)) {
        concretes.add(displayName);
    }
}

} else {
    e = (Element) initialState;

    Q.add(initialState);
    displayName = e.getAttribute("DisplayName").replace(' ', '_');
    XNameToDisplayName.put(e.getAttribute("x:Name"), displayName);
    if (foundations == null || !foundations.contains(displayName)) {
        concretes.add(displayName);
    }
    nextState = e.getAttribute("DisplayName").replace(' ', '_');
    outputStrBuffer.append(currState);
    outputStrBuffer.append(" -> ");
    outputStrBuffer.append(nextState);
    outputStrBuffer.append(" + ");
}

while (Q.peek() != null) {
    node = Q.remove();
    e = (Element) node;

    currState = e.getAttribute("DisplayName").replace(' ', '_');
    transitions = (NodeList) xpath.compile(
        "./State.Transitions/Transition").evaluate(node,
        XPathConstants.NODESET);

    for (int i = 0; i < transitions.getLength(); i++) {

        // back/cross edges
        // attribute "To" exists in the "Transition" node
        e = (Element) transitions.item(i);
        String attr = e.getAttribute("To");
        if (attr != null && !attr.isEmpty()
            && attr.trim().length() > 0) {
            String[] strArray = attr.split(" ");
            String xName = strArray[1].substring(0,
                strArray[1].length() - 1);
            nextState = XNameToDisplayName.get(xName);
            if (nextState != null) {
                outputStrBuffer.append(currState);
                outputStrBuffer.append(" -> ");
            }
        }
    }
}

```

```

        outputStrBuffer.append(nextState);
        outputStrBuffer.append(" + ");
    } else {
        // edges not yet discovered
        delayedEdges.put(xName, currState);
    }
    continue;
}

// tree edges
node = (Node) xpath.compile("./Transition.To/State")
    .evaluate(transitions.item(i), XPathConstants.NODE);
if (node != null) {
    e = (Element) node;

    Q.add(node);
    displayName = e.getAttribute("DisplayName").replace(
        ' ', '_');
    XNameToDisplayName.put(e.getAttribute("x:Name"),
        displayName);
    if (foundations == null
        || !foundations.contains(displayName)) {
        concretes.add(displayName);
    }

    nextState = e.getAttribute("DisplayName").replace(' ',
        '_');
    outputStrBuffer.append(currState);
    outputStrBuffer.append(" -> ");
    outputStrBuffer.append(nextState);
    outputStrBuffer.append(" + ");

    // if nextState is a final state, append
    // nextState + " -> End + "
    String IsFinal = e.getAttribute("IsFinal");
    if (IsFinal != null && IsFinal.equals("True")) {
        outputStrBuffer.append(nextState);
        outputStrBuffer.append(" -> ");
        outputStrBuffer.append("End");
        outputStrBuffer.append(" + ");
    }
} else {

    // back/cross edges
    // the "Transition.To" node is under the "Transition"
    // node
    node = (Node) xpath.compile("./Transition.To/*")
        .evaluate(transitions.item(i),
            XPathConstants.NODE);
    e = (Element) node;

    if (node == null) {

        System.err.println("currState: " + currState);
        System.err.println("node is null");
    }
}

```

```

    }
    nextState = XNameToDisplayName.get(e.getTextContent());
    if (nextState != null) {
        outputStrBuffer.append(currState);
        outputStrBuffer.append(" -> ");
        outputStrBuffer.append(nextState);
        outputStrBuffer.append(" + ");
    } else {
        // edges not yet discovered
        delayedEdges.put(e.getTextContent(), currState);
    }
}
}

// print back/cross edges that has not yet been discovered when
// first seen
for (String key : delayedEdges.keySet()) {
    outputStrBuffer.append(delayedEdges.get(key));
    outputStrBuffer.append(" -> ");
    outputStrBuffer.append(XNameToDisplayName.get(key));
    outputStrBuffer.append(" + ");
}

// remove " + " at the end of the StringBuffer
if (outputStrBuffer.length() >= 3
    && outputStrBuffer.charAt(outputStrBuffer.length() - 1) == ' '
    && outputStrBuffer.charAt(outputStrBuffer.length() - 2) == '+'
    && outputStrBuffer.charAt(outputStrBuffer.length() - 3) == ' ') {
    outputStrBuffer.setLength(outputStrBuffer.length() - 3);
}

return outputStrBuffer.toString();
}

public static void main(String[] args) {
    try {
        DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory
            .newInstance();
        DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();

        if (args == null || args.length < 2) {
            System.err
                .println("Error: need the pathnames of 2 .xml file as arguments");
            return;
        }

        Document doc1 = docBuilder.parse(new File(args[0]));
        Document doc2 = docBuilder.parse(new File(args[1]));

        HashSet<String> foundations = new HashSet<String>();

```

```

HashSet<String> concretes = new HashSet<String>();

String w1 = getAlloyEdges(doc1, null, foundations);
String w2 = getAlloyEdges(doc2, foundations, concretes);

// produce translator.thm
Collection<String> allNodes = new LinkedList<String>();
allNodes.addAll(foundations);
allNodes.addAll(concretes);
produceThm(new PrintWriter(System.err), allNodes);
// produceThm("translator.thm", allNodes);
// System.err.println("test to stderr");

System.out.println();
if (foundations.size() > 0) {
    System.out.print("one sig ");

    boolean isFirst = true;
    for (String DisplayName : foundations) {
        if (!isFirst) {
            System.out.print(", ");
        } else {
            isFirst = false;
        }
        System.out.print(DisplayName);
    }

    System.out.println(" extends Foundation {}\n");
}
if (concretes.size() > 0) {
    System.out.print("one sig ");

    boolean isFirst = true;
    for (String DisplayName : concretes) {
        if (!isFirst) {
            System.out.print(", ");
        } else {
            isFirst = false;
        }
        System.out.print(DisplayName);
    }

    System.out.println(" extends Concrete {}\n");
}
System.out.println("fact W1defn {}");
System.out.print("\tw = {}");
System.out.print(w1);
System.out.println("{}");
System.out.println("{}");
System.out.println();
System.out.println("fact W2defn {}");
System.out.print("\tv = {}");
System.out.print(w2);
System.out.println("{}");
System.out.println("{}");

```



```
    } catch (Exception e) {  
        System.err.println("Error: " + e.getMessage());  
        e.printStackTrace(new PrintStream(System.out));  
    }  
}  
  
}
```

Appendix G

Automator.java Documentation

main

public static void main(String[] args)
throws Err

The main method takes the file names of an als file (contains the original workflow and the derived workflows) and a thm file. It then produces a dot file that contains the graphical representation of the workflows within the als file in the format specifiers in the thm file.

The name of the produced dot file is based on the name of the als file.

For example, if the als file is called **RFI-Conformance.als**, the dot file will be named **RFI-Conformance.dot**.

note:

- als file is the file format used by alloy.
- A thm file specifies the format of graphical representation used by an als file.

Parameters:

- args[0] - the path to the als file that contains the original workflow and the derived workflow.
- args[1] - the path to the thm file that will be used by args[0].

Returns:

- void

Side Effect:

- produces a dot file that contains the graphical representation of the workflows within the als file in the format specifiers in the thm file.

Appendix H

Automator.java Code

```
/* Alloy Analyzer 4 -- Copyright (c) 2006-2009, Felix Chang
*
* Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation
files
* (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT
LIMITED TO THE WARRANTIES
* OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR COPYRIGHT HOLDERS BE
* LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

import java.io.IOException;
import java.io.File;
import java.io.PrintWriter;

import edu.mit.csail.sdg.alloy4.A4Reporter;
import edu.mit.csail.sdg.alloy4.Err;
import edu.mit.csail.sdg.alloy4.ErrorWarning;
import edu.mit.csail.sdg.alloy4compiler.ast.Command;
import edu.mit.csail.sdg.alloy4compiler.ast.Module;
import edu.mit.csail.sdg.alloy4compiler.parser.CompUtil;
import edu.mit.csail.sdg.alloy4compiler.translator.A4Options;
import edu.mit.csail.sdg.alloy4compiler.translator.A4Solution;
import edu.mit.csail.sdg.alloy4compiler.translator.TranslateAlloyToKodkod;
import edu.mit.csail.sdg.alloy4viz.VizGUI;
import edu.mit.csail.sdg.alloy4viz.VizState;

/**
 * Run Alloy commands in als file, then visualize and apply
 * appropriately named thm file, saving output in Graphviz/Dot format. */

public class Automator {

    /*
     * Execute every command in every file.
     *
     * This method parses every file, then execute every command.
     *
     * If there are syntax or type errors, it may throw
     * a ErrorSyntax or ErrorType or ErrorAPI or ErrorFatal exception.
     * You should catch them and display them,
```

```

* and they may contain filename/line/column information.
*/
public static void main(String[] args) throws Err {

    // The visualizer (We will initialize it to nonnull when we visualize an Alloy solution)
    VizGUI viz = null;

    // Alloy4 sends diagnostic messages and progress reports to the A4Reporter.
    // By default, the A4Reporter ignores all these events (but you can extend the A4Reporter to display the event for the user)
    A4Reporter rep = new A4Reporter() {
        // For example, here we choose to display each "warning" by printing it to System.out
        @Override public void warning(ErrorWarning msg) {
            System.err.println("Relevance Warning:\n"+(msg.toString().trim()+"\n\n"));
            System.err.flush();
        }
    };

    if(args == null || args.length < 1){
        System.exit(0);
    }
    // loop over every als file named on the command line
    //for(final String filename:args) {

    String filename = args[0];
    // Parse+typecheck the model
    // System.out.println("==== Parsing+Typechecking "+filename+" =====");
    Module world = CompUtil.parseEverything_fromFile(rep, null, filename);

    // Choose some default options for how you want to execute the commands
    A4Options options = new A4Options();

    options.solver = A4Options.SatSolver.SAT4J;

    for (Command command: world.getAllCommands()) {
        // Execute the command
        // System.out.println("==== Command "+command+": =====");
        A4Solution ans = TranslateAlloyToKodkod.execute_command(rep, world.getAllReachableSigs(), command, options);

        // Print the outcome
        // System.out.println(ans);

        // If satisfiable...
        if (ans.satisfiable()) {
            // You can query "ans" to find out the values of each set or type.
            // This can be useful for debugging.
            //
            // You can also write the outcome to an XML file
            ans.writeXML("alloy_example_output.xml", world.getAllFunc());
            //
            // You can then visualize the XML file by calling this:
            if (viz==null) {
                viz = new VizGUI(false, "alloy_example_output.xml", null);
                // System.out.println("new viz");
            } else {
                // viz.loadXML("alloy_example_output.xml", true);
            }
        }
    }
}

```

```

    // System.out.println("old viz");
}
VizState vs = viz.getVizState();
final String tn;
if(args.length >= 2){
    tn= args[1];
} else {
    tn = filename.replace(".als", ".thm");
}
try{
    final File f = new File(tn);
    final String s = f.getCanonicalPath();
    vs.loadPaletteXML(s);
} catch(IOException e){
    System.err.println("Error: cannot find/read " + tn);
    System.err.println(e.getMessage());
    System.err.println(e.getStackTrace());
    return;
}
viz.loadXML("alloy_example_output.xml", true);

// write dot output
final String dn;
if(args.length >= 3){
    dn = args[2];
} else {
    dn = filename.replace(".als", ".dot");
}
try{
    final PrintWriter w = new PrintWriter(new File(dn));
    w.print(viz.getViewer().toString());
    w.close();
} catch(IOException e){
    System.err.println("Error: cannot find/read " + tn);
    System.err.println(e.getMessage());
    System.err.println(e.getStackTrace());
    return;
}

// we will only execute the first command, then we exit
System.exit(0);
//}
}
}
}
}

```