

THESES & DISSERTATION
SECTION
CENTRAL LIBRARY, T.U.

CENTRAL LIBRARY
TEZPUR UNIVERSITY

Accession No. T 262

Date 15/1/14

A Pattern-Based Approach For Maintaining
A Constrained Workflow

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

by

Achyanta Kumar Sarmah, Registration No. 009 of 2011



School of Engineering
Department of Computer Science and Engineering
Tezpur University
Napaam-784028, Assam, India
November, 2013

THESES & DISSERTATION
SECTION
CENTRAL LIBRARY, T.U.

*In fond memory of Jyoti da, Juli bou and Jiri...
you will always be there in our hearts*

Abstract

Workflow is a process-centric domain. Composition of a workflow is done on the basis of the functional and non-functional characteristics. Functional characteristics, like ordering of tasks, execution rules, roles etc. give rise to Workflow Patterns (WPs). Non-functional characteristics, like security, performance etc. need to be integrated to the workflow for its robustness. It is possible to identify patterns in a given workflow. WPs in a workflow are related to one another. In practice, both run-time and design-time changes may occur in these relationships after the initial enactment. Run-time changes could be raised by the roles executing the tasks within patterns. Design-time changes could occur due to changes in user specifications. It is the concern of the workflow composer to gather the knowledge of the workflow from user specifications and give a consistent representation. Articulating this in terms of patterns and their relationships would make the task easier for the composer in two ways - firstly, considering workflow at pattern level allows to hide the complexity of the workflow; secondly, it reduces the work to be done for the verification of changes to be introduced. Non-functional characteristics can be integrated into a workflow in two ways - during the composition of the workflow or at a later stage as and when the need arises. For evolution of a workflow as time progresses, it needs to be maintained by incorporating changes in such a manner that the relationships amongst the existing patterns and the new patterns remain consistent. By consistency, we mean that the cascading effect of these changes gets propagated duly through the whole of the workflow.

The work reported in this thesis proposes a unique pattern-based ap-

proach for composing and maintaining a workflow. An architecture of the proposed approach is given which consist of three modules - (1) Formal organization of patterns as lattices; (2) Composition of workflow as a directed graph of patterns and temporal constraints amongst them; (3) Incorporation of changes into the workflow in a consistent manner. Procedures based upon tools and algorithms from Formal Concept Analysis (FCA) are devised for generating pattern lattices. For securing the composed workflow, a trust-based approach has been taken instead of the usual threat-based approach. Security solutions are represented as Security Patterns (SP). An SP is defined in terms of tasks to be performed. This enables integrating SPs to WPs resulting in a formal secured version of a WP viz: Secured Workflow Pattern (SWP).

For incorporating changes into the composed workflow, WPs are considered as Reference Intervals from Allen's Interval Algebra (IA) where Reference Interval is a cluster of time intervals where the temporal constraints amongst the intervals are fully computed. Workflow being a process-centric representation, the constituent WPs within a workflow needs to be viewed as a hierarchy of sub-processes. The temporal constraints amongst sub-processes within a workflow are specified in terms of some reference system instead of absolute temporal terms. A hierarchy of reference intervals viz: Reference Interval Hierarchy (RIH) is traced from the workflow graph. Change incorporation is achieved by three functionalities - a transform function that traces a RIH from the workflow graph; a constraint propagation function that incorporates raised changes into the RIH; an inverse transform function that updates the corresponding workflow graph with the changes in the RIH.

For representation of the interacting roles, a Role Enabling Base(REB) from the Temporal Role Based Access Control (TRBAC) framework is attached with each WP in the workflow. An REB is a construct that allows us to specify the roles attached to a WP and their enabled and disabled status during an execution period of the WP. The role to task assignment for a WP is maintained as a mapping from the set of tasks constituting the WP and the set of roles attached with it.

The main contribution of this research work is the unique pattern-based approach for composing and maintaining a workflow in a robust manner. A modular architecture for the approach has been given with the following components -

- **Concept Lattice generation and navigation** - A *LatticeGenerator* procedure has been devised which generates the concept lattice while preserving the sub-concept and super-concept relationships along with the generated concepts. A *LatticeNavigator* procedure is devised which navigates through the concept lattice and returns the desired concept.
- **Workflow Pattern Lattice (WPL) and Security Pattern Lattice (SPL)** - WPs and SPs are formalized based on concepts from FCA and organized into concept lattices. Such a formal approach to organization of patterns hasn't been attempted before.
- **Workflow composition**- Considering temporal constraints amongst patterns within a workflow, a composition procedure has been devised that outputs a workflow from user specifications in the form of a directed graph where vertices are the patterns and edges are the temporal

constraints amongst them. The composed workflow is secured at the pattern level where the SPs are integrated to the constituent WPs. SPs are defined as tasks to be executed instead of some metric to be achieved as is usually done. This enables a WP to be integrated with SPs in such a manner that the resultant Secured Workflow Pattern(SWP) essentially remains a WP.

- **Change incorporation procedure** - Change incorporation into the composed workflow has been achieved by three functionalities
 - *Transformation of a workflow graph to RIH* The directed graph of the workflow is transformed to an equivalent RIH in Allen's IA. Such a representation of workflow in IA hasn't been attempted before.
 - *Change incorporation* - Making use of a path-consistent procedure, changes are incorporated into the RIH such that the cascading effect of the change is taken care of properly.
 - *Inverse transformation of RIH to workflow graph*- Once the changes are incorporated into the RIH, the updated RIH is transformed back into it's directed graph form

The overall approach proposed in this thesis serves as a foundation for re-engineering workflow of different domains in terms of patterns and constraints.

Keywords — *Workflow, Workflow Pattern, Security Pattern, Secured Workflow Pattern, Temporal Role Based Access Control, Formal Concept*

Analysis, Concept Lattice, Workflow Pattern Lattice, Security Pattern Lattice, Allen's Interval Algebra



TEZPUR UNIVERSITY

Certificate

This is to certify that the thesis titled "A Pattern-Based Approach For Maintaining A Constrained Workflow" submitted to Tezpur University in the Department of Computer Science and Engineering under the School of Engineering in partial fulfilment of the award of the degree of Doctor of Philosophy in Computer Science and Engineering is a record of research work carried out by Mr. Achyanta Kumar Sarmah under our supervision and guidance. All helps received by him from various sources have been duly acknowledged. No part of this thesis has been submitted elsewhere for award of any other degree.

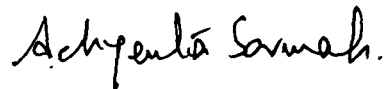
Signature of Supervisor
(Smriti Kumar Sinha)
Professor,
Computer Science and Engineering

Signature of Co-Supervisor
(Shyamanta Moni Hazarika)
Professor,
Computer Science and Engineering

School of Engineering
Tezpur University

Declaration

I, Achyanta Kumar Sarmah, hereby declare that the thesis entitled "*A Pattern Based Approach For Maintaining A Constrained Workflow*" submitted to the Department of Computer Science and Engineering under the School of Engineering, Tezpur University, in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy, is based on bona fide work carried out by me. The results embodied in this thesis have not been submitted in part or in full, to any other university or institute for award of any degree or diploma.



(Achyanta Kumar Sarmah)

Acknowledgments

I take this opportunity to express my gratitude to Prof. Smriti Sinha for his continual trust on my capabilities and accepting me as his scholar. I express my gratitude towards Prof. Shyamanta Hazarika for accepting me as his scholar and supporting me throughout this research work. Dear Sirs, you remain the constant source of inspiration in my life. I also take this opportunity to express my gratitude towards my teachers in the Dept. of Computer Science and Engineering for their continual support and empathy. My friends in the department, what would I have done without your care? Thank you so much.

Kastun, no words find place to articulate your contribution in whatever I have achieved here. You understand me each moment and have stood by my side allowing me to work ahead with my research while shouldering all my responsibilities. I express my regards to my elders standing by my side always and my love towards my younger for having the faith in me.

Anjub and Ajay, you have been there encouraging me towards achieving my dream of higher studies each and every moment.

I take this opportunity to express my gratitude towards Tezpur University for allowing me the space and time to carry forward my PhD to completion. My sincere thanks remain towards one and all in IIM Shillong for giving me the scope and all the encouragement to carry on my research to completion even during the toughest of time.

Thank you 'Cygwin' and 'Netbeans' you provided the platform independence for this research!

Last but not the least, my thanks remains towards all those who have been instrumental in the completion of my thesis but whose help have escaped my remembrance.

Without you all, this thesis wouldn't have seen the light of the day.

(Achyanta Kumar Sarmah)

Contents

1	Introduction	2
1.1	Genesis of the Research	2
1.2	Issues to be Addressed	6
1.3	Scope of the Research	8
1.4	Organization of the Thesis	8
2	Review Work	12
2.1	Formal Concept Analysis	12
2.1.1	Theoretical Background	12
2.1.2	Research and Usage of FCA	16
2.1.3	Existing Concept Lattice Generating Algorithms	19
	Ganter's Next-Closure Algorithm	20
2.2	Workflow and Workflow Pattern	21
2.2.1	Van Der Aalst <i>et al</i> Repository of WP	24
2.3	Security and Security Pattern	26
2.3.1	Repository of Security Patterns	26

2 4	Allen's Interval Algebra	28
2 5	Temporal Role Based Access Control	31
3	An Architecture of the proposed approach	36
3 1	The Architecture	36
3 1 1	Formal Organization of Patterns as Lattice	38
3 1 2	Workflow Composition as Directed Graph	39
3 1 3	Change Incorporation in a Workflow	40
3 2	Conceptual Layers of the Architecture	41
3 3	Enactment of the Architecture	45
4	Formal Organization of Patterns	48
4 1	Formal Organization of Workflow Patterns	49
4 1 1	Workflow Pattern Lattice	50
4 1 2	Basic Formalization	50
4 2	Formal Organization of Security Patterns	54
4 2 1	Security Pattern Lattice	55
4 2 2	Basic Formalization	56
4 3	Generating Concept Lattice	58
4 3 1	Concept Lattice Generating Procedure	58
4 3 2	Generating the Workflow Pattern Lattice	63
4 3 3	Generating the Security Pattern Lattice	67
4 4	Navigating in Pattern Lattices	69

4 4 1	Scope of Navigation	69
4 4 2	Navigation Criteria	69
4 4 3	Navigation Procedure	70
5	Composition of a Workflow Graph	77
5 1	Composition without Security	81
5 1 1	Constraint Along a Walk	81
5 1 2	Validating a Constraint	82
5 1 3	The Composition Procedure	84
5 1 4	Illustrative Example without Security	87
5 2	Composition with Security	89
5 2 1	Secured Workflow Pattern	90
5 2 2	Securing a Workflow	90
5 2 3	Illustrative Example with Security	92
6	Incorporating Changes in a Workflow	96
6 1	Basic Formalization	97
6 2	Functionalities Involved	103
6 2 1	Building the <i>Reference Interval Hierarchy</i>	104
6 2 2	Incorporating Change	107
6 2 3	Inverse Transforming <i>Reference Interval Hierarchy</i> to Workflow Graph	110
6 3	Change in Roles Arising from Changes in Workflow	113

6 3 1	Introduction of a New Role	114
6 3 2	Reassignment of an Existing Role	114
6 4	Change in Security Arising from Changes in Workflow	115
6 4 1	Introduction of a New Security Pattern	116
6 4 2	Change in Existing Constraint	116
6 5	Illustrating the Approach	117
6 5 1	Illustrating Workflow Pattern	117
6 5 2	Illustrating Changes Introduced in Workflow	119
	Introduction of a New Pattern	119
	Change in Value of an Existing Constraint	120
6 5 3	Illustrating Changes on Roles	123
	New Roles Introduced with a New Pattern	123
	Existing Roles Assigned to a New Pattern	124
7	Summary and Future Directions	126
7 1	Summary of the Work Done	126
7 1 1	Contributions	127
7 1 2	Algorithms presented	130
7 2	Analysis and Future Work	132
A	FCA Usage and Tools Developed	135
B	Workflow Tools and Deliverables	141

C Security Patterns Usage	145
D WP and WC Enumeration	154
E SP and TE Enumeration	158
F Patterns borrowed from Van Der Aalst's repository	161

List of symbols

G	Directed Graph, Definition 25	16
E_i	Event expression, Definition 7	32
C_i	Role status expression Definition 7	32
$p \in E$	Prioritized event, Definition 7	32
\prec	Priority, Definition 7	32
WF	Workflow, Definition 11	51
S	Start pattern in a workflow Definition 11	51
P	Patterns in a workflow Definition 11	51
F	Finish patterns in a Workflow, Definition 11	51
C	Constraints in a Workflow, Definition 11	51
R	Roles in a Workflow, Definition 11	51
τ	Mapping from role to task, Definition 12	51
V_{index}	Index of WP, Definition 12	51
CFrules	Execution rules of a WP Definition 12	51
G_w	WPs in a workflow context, Definition 14	53
M_w	WCs in a workflow context Definition 14	53
I_w	Mappings in a workflow context, Definition 14	53
\mathcal{W}	Workflow context, Definition 16	53

\bar{W}	WPL Definition 16	53
G_s	SPs in a security context, Definition 19	57
M_s	TEs in a security context Definition 19	57
I_s	Mappings in a security context, Definition 19	57
S	Security context, Definition 21	57
\bar{S}	SPL, Definition 21	57
G_{mc}	Objects in a Multi-Context Definition 22	71
A_{mc}	Attributes in a Multi-Context Definition 22	71
I_{mc}	Relations in a Multi-Context , Definition 22	71
G_i	Objects in a Multi-Concept Definition 23	71
A_i	Attributes in a Multi-Concept Definition 23	71
R_i	Related objects Definition 23	71
G_{ms}	SPs in a Multi-Context SP Definition 24	74
A_{ms}	Preconditions Definition 24	74
I_{ms}	Relationships in a Multi-Context SP, Definition	
	24	74
G	Directed graph Definition 25	77
ρ	Temporal constraints in WF, Proposition 1	78
σ	Execution Walk, Definition 27	97

List of Figures

2.1	Integer Concept lattice	16
2.2	Usage of FCA	19
2.3	Area wise publications in workflow	23
2.4	Area wise publication in WP	24
3.1	Architecture of the proposed approach	37
3.2	Pattern Lattice generation	39
3.3	Workflow composition	40
3.4	Change incorporation	41
3.5	Conceptual layers	42
3.6	Enactment of the architecture	46
4.1	Trust-Based Security Model	55
4.2	Flow of steps in generation of WPL	64
4.3	Workflow Pattern Lattice	66
4.4	Security Pattern Lattice	68

5 1	Requisition processing workflow	89
5 2	Security constraints	92
5 3	Secured Sequenced pattern	94
5 4	Secured Requisition processing workflow	95
6 1	Basis of proof for Proposition 2	99
6 2	Transform of basis case of Proposition 2	100
6 3	Case 1 of proof for Proposition 2	100
6 4	Transformation of Case 1 for Proposition 2	101
6 5	Case 2 of proof for Proposition 2	101
6 6	Transformation of Case 2 for Proposition 2	103
6 7	Role change scenario	113
6 8	Interval hierarchy of the requisition processing workflow	118
6 9	Interval hierarchy of Exclusivechoice2	119
6 10	Introducing a new pattern	121
F 1	Sequence Pattern	162
F 2	Exclusive Choice Pattern	162
F 3	Parallel Split Pattern	163
F 4	Structured Loop (while) Pattern	164
F 5	Structured Loop (repeat) Pattern	164

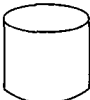

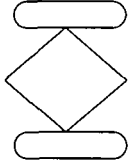

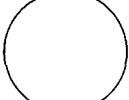

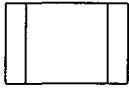
List of Tables

2.1	Allen's 13 interval relations	29
2.2	Transitivity table for the 12 temporal relations in Allen's IA omitting '='	30
4.1	Workflow Pattern Context	65
4.2	Security Context	68
5.1	WPs' roles and constraints in requisition processing workflow	87
5.3	Security requirements of Requisition Processing workflow	93
5.4	Secured Sequence1	93
6.1	Consistency check table for role triggers in REB	115
6.2	Exclusivechoice2	119
6.3	Sequence4	120

List of Algorithms

1	Calculate Transitive Constraint	31
2	LatticeGenerator (<i>SingleAttributeConcepts, MaxConcepts, MinConcepts</i>)	62
3	LatticeNavigator (<i>ConceptLattice, RequiredAttributes, MaxConcept, MinConcept</i>)	72
4	MultiContextNavigator (Rel_Obj, Vect_Selected_Obj, No_Atr, Atr)	73
5	ConstraintAlongWalk (A,B,Cons)	83
6	ValidateConstraint (p_a, p_b, Cons)	84
7	ComposeWG (SelfPatSet, Cons)	86
8	Exploresubgraph ($P_{cur}, W_{pat}, \text{Cons}, RIH_{pat}, \text{TCons}$)	106
9	Transform (W_{pat}, Cons)	107
10	IntroduceNewPattern ($Pat_a, Pat_a, Pat_{new}, C_1, C_2, RIH_{pat}, \text{TCons}$)	109
11	ChangeConstraint ($C_{new}, Pat_a, Pat_b, RIH_{pat}, \text{TCons}$)	110
12	InverseTransform (WF, RIH_{pat}, TCons, W_{pat}, Cons)	112

List of Legends

	Database	37
	Interacting entity	37
	Formal concept lattice	37
	File	37
	Function	37
	Rule set	63
	Algorithm	63

List of abbreviation

FCA	Formal Concept Analysis	5
WP	Workflow Pattern	5
WPL	Workflow Pattern Lattice	5
SP	Security Pattern	5
SPL	Security Pattern Lattice	5
RBAC	Role Based Access Control	6
TRBAC	Temporal Role Based Access Control	6
REB	Role Enabling Base	6
IA	Interval Algebra	6
RI	Reference Interval	6
RIH	Reference Interval Hierarchy	6
WC	Workflow Concern	10
TE	Trust Element	10
MAC	Maximum Allowable Constraint	84
SWP	Secured Workflow Pattern	89

Chapter 1

Introduction

Process-centric technologies like workflow have undergone a sea change over the years. Advancements in different sectors have brought in a proliferation of specialized processes. With passage of time, human tendency has been to attain a higher degree of formalization in processes. Formalization brings with it an ability to automate, and together with it the ability for re-engineering and reuse. This is important as people tend to revise the way in which work has been done in the past, reuse most of the basic tasks but with lesser effort.

1.1 Genesis of the Research

Let us have a look at how processing of a requisition could happen in an organization. A typical scenario is stated below in Example 1.

Example 1 *A requisition can be **created** by an employee with rights as-*

signed. Once created, it is delivered to a Store Officer (SO) who pushes it to the **Approve Requisition** process. On rejection, the requester is notified and the process terminates. On acceptance, the requisition is archived and SO executes **Generate Purchase Order** process. The Purchase Order (PO), casted with the approved requisition, is pushed to the **Approve PO** process by SO. There after, on rejection, SO is notified and the process terminates. On acceptance, the casted PO is archived and SO executes **Dispatch PO** process. On **Delivery to store**, Store Keeper (SK) executes **Generate Receipt** process. The Receipt is casted with the PO and **Update Stock** process is executed by SK. The casted Receipt is then archived and delivered to Accounts Officer (AO). AO executes **Generate Voucher** process and pushes the voucher to the **Accounts Verification** process. On being verified and approved, the Voucher is casted with the Receipt. Then the processes **Create Payment** and **Dispatch Payment** are executed by the AO and the overall requisition processing terminates. On rejection, the voucher is delivered back to AO who creates the voucher afresh and pushes it again to **Accounts Verification**. This process iterates until the voucher is accepted. This is because a voucher for a delivered PO cannot be rejected indefinitely. Processes in this workflow are temporally constrained - PO approval starts **after** PO generation, item delivery starts after PO dispatch, Update stock and voucher generation can start **together**, Approvals could be done only **during** 1st of April to 30th of March of each year etc. An employee can raise requisition only after credentials verification. Changes in the form of periodic enabling and disabling of processes and roles were introduced in due course of time.

Observably, subsets of the processes form similar type of patterns at different stages - A sequence is formed out of 'Create Requisition' and 'Approve Requisition'; another sequence is formed of 'Delivery to store' and 'Generate Receipt', so on and so forth. Execution order of the processes are seen to be temporally constrained - 'Generate PO' can start only after 'Approve Requisition' etc. Thus it is realized that formalizing these patterns structurally and consolidating them into some formal organization would actually make the task of re-engineering and reuse of the processes much easier. It could be seen that the patterns would follow different walks of executions depending on how the control-flow rules are satisfied - In 'Approve Requisition' process, if the requisition is approved 'Generate PO' is executed and the processing continues. If it is rejected the processing of the requisition terminates with 'Notify Requester'. Also, the execution order of the patterns in these walks are temporally constrained - The 'ExclusiveChoice' formed of { 'Approve PO', 'Notify Store Keeper', 'Dispatch PO' } can 'start' only after Sequence of { 'Create Requisition', 'Approve Requisition' }. Thus the whole of the procedure could in fact be composed as a directed graph of the patterns and their temporal constraints. In addition to this, changes are seen to be incorporated into the workflow from time to time so that it evolves robustly.

Considering this motivating scenario as an ideal, could we find a common and convenient way of re-engineering workflows from various domains and platforms ? This very question leads us to the genesis of this research -

Achieving a formal approach that considers a workflow as a graph of it's constituent patterns, increase it's robustness by incorporating non-functional characteristics like security and facilitate it's

evolution over passage of time by incorporating changes into it

A formal architecture for workflow composition including incorporation of changes is hitherto found to be missing. Most of the existing workflow technologies available are context-specific. However the underlying processes remain same. The major thrust of this work has been to provide a formal structure for such a scenario based on 'patterns'

Without a formal structure to base upon, it is increasingly difficult to re-engineer workflows. Re-engineering would enable us to keep pace with the changes happening around. In this thesis a new pattern-based approach for design of workflow that brings a formal character to workflow composition through the use of concepts from Formal Concept Analysis (FCA) is proposed. This formal architecture of workflow in terms of patterns and constraints is the prime contribution of this work. Composition of a workflow is done on the basis of the functional and non-functional characteristics of the context. Functional characteristics like ordering of tasks, execution rules, executing roles etc. give rise to Workflow Patterns (WPs). Non-functional characteristics like security need to be integrated to the workflow for its robustness. Security needs could be integrated as Security Patterns (SPs) to the context where required. Run-time and design-time changes occur to the relationships amongst patterns within a workflow. These WPs and SPs mined from the literature are organized into two Formal Concept Lattice [1] viz. Workflow Pattern Lattice (WPL) and Security Pattern Lattice (SPL).

It is the concern of the workflow composer to gather the knowledge of the workflow from user specification and give a consistent composition. Articulating this in terms of patterns and their relationships would make the task

easier for the composer in two ways - firstly, considering workflow at pattern level allows hiding the complexity of the workflow, secondly it reduces the work to be done for the verification of changes to be introduced. For achieving this a workflow is composed as a directed graph of it's patterns drawn from the WPL and SPL and the constraints amongst them. For representation of the interacting roles a Role Enabling Base (REB) from the Temporal Role Based Access Control (TRBAC) [2] framework is attached with each WP in the workflow. An REB is a construct for specifying the roles attached to a WP and their enabled and disabled status during an execution of the WP.

Change is inevitable in any system executing in an interacting environment and a workflow is no exception to it. A workflow would evolve with incorporation of changes that keeps on arising over passage of time. For facilitating this, the workflow graph is transformed into a hierarchy of Reference Intervals (RI) from Allen's Interval Algebra (IA) [3] viz. Reference Interval Hierarchy (RIH), changes are incorporated into the RIH in a path consistent manner and the updated RIH is transformed back to it's directed graph form. The requisition processing workflow given in Example 1 is used to illustrate the proposed approach through out this thesis.

1.2 Issues to be Addressed

A formal approach to composition and maintenance of a workflow in terms of it's constituent patterns involves issues as stated below.

Organizing patterns into a formal structure - Workflow as a process technology has been around for quite some time now. Research and development in this area has actually resulted in a corpus of WPs as is revealed in the review work in Chapter 2. A formal approach for organizing these patterns would help in consolidating them in one platform and serve as a central repository for retrieval of existing and addition of newer patterns. Issues involved for such an approach are - Formal definitions of workflow constructs, Pattern repository, Generating procedure for the organization of patterns, Navigating procedure into the organization of patterns.

Workflow composition in terms of patterns and constraints - Users in general give specification of workflow in their own way which are usually unstructured. For a formal approach these specifications need to be structured and formalized. Composition with these formalized specifications needs to be done in such a manner that the composed workflow finally remains consistent in terms of the constraints. Hence verifying that a constraint being added does not make the already composed part of the workflow inconsistent is required. A workflow thus composed could follow different paths of execution. Thus for a particular enactment, one needs to have a way of marking the workflow. Non-functional characteristics like security also need to be integrated to the workflow for making it robust. Thus the issues in composition of a workflow are - Formalization of user specification, Constraint verification, Marking workflow and Integrating security.

Incorporating changes - A workflow is an interactive system that evolves over passage of time. The changes arise from time to time due to changing needs of users and roles. Run-time changes could be done by the roles executing the tasks within patterns. Design-time changes could occur due to changes in user specifications. These changes need to be incorporated in such a manner that the consistency of the workflow is maintained during any execution of the workflow and structurally the workflow remains robust.

1.3 Scope of the Research

The scope of this research is illustrated in the architecture given in Chapter 3. It addresses the issues involved in three modules - Formal organization of patterns, Workflow composition, Change incorporation. For formal organization of patterns, the scope includes Van Der Aalst *et al*'s repository of workflow patterns for functional characteristics and security patterns for non-functional characteristics. Workflow composition is done in the form of a directed graph of temporal constraints amongst the constituent patterns of the workflow. Change incorporation is done in terms of a new pattern being introduced and existing patterns being changed.

1.4 Organization of the Thesis

The remaining of this thesis is organized as follows

Chapter 2-

This chapter presents an in-depth review work primarily keeping in mind resolution of the issues involved. FCA was reviewed in search of a formal organizing tool for patterns. Thus Formal Concept Lattice from FCA framework has been used for constructing formal organization of WPs and SPs. Review work on workflow and workflow patterns was done to explore for repositories of WPs and some architecture of constructing and maintaining a pattern-based workflow. Temporal constraints amongst WPs have been considered for composing workflow. In this regard Allen's Interval Algebra (IA) was reviewed for ways and formalisms of representing a workflow and its patterns in terms of time intervals and relationships amongst them. In order to formulate access control of the workflow via roles, RBAC was reviewed and the RBAC construct from the same was used for periodic enabling and disabling of roles attached with WPs. Review work on security was done for repositories of SPs and formal organization of the same. Though many repositories in varied contexts were found to be existing, a formal organization of the same was found to be missing and this forms a part of this research.

Chapter 3- In this chapter, an architecture of the proposed approach for composing and maintaining a pattern-based workflow is given. In addition to this, the conceptual layers and an enactment cycle of the architecture that gives the flow of execution of the steps involved in enactment of the proposed approach are given.

Chapter 4- In this chapter, an approach for a formal organization of pat-

teins has been achieved in the form of a formal lattice. WPL has been achieved from the Van Der Aalst's repository of WPs which are characterized by Workflow Concerns (WC). The SPL has been generated from the existing repository of SPs. Here a trust-based approach has been taken and SPs has been characterized by Trust Elements (TE). In addition to this, procedures for generating and navigating in the lattices based on Gantner Next-Closure algorithm has also been devised.

Chapter 5- This chapter achieves composition of a workflow as a directed graph of its patterns and temporal constraints. For constraints, it considers the set of 13 temporal relationships between a pair of ordered intervals as established in Allen's IA. The approach also secures a workflow by integrating SPs to WPs such that the resultant pattern essentially remains a WP. In addition to this, the proposition 1 has been proved that establishes the feasibility of the approach given.

Chapter 6- In this Chapter, an innovative approach has been taken for incorporating changes in a workflow composed as a directed graph of its patterns and temporal constraints amongst them. For the first time, the concept of Reference Interval Hierarchy based on the Reference Interval construct of Allen's IA has been formalized. Establishing the fact that a workflow representable as a directed graph can be transformed into a RIH, the approach works out three functionalities for incorporating changes - A transform function that traces a RIH from a workflow W . A constraint propagation function that incorporates changes into the RIH. An inverse transform function that transforms back the up-

dated RIH to it's directed graph form. These functionalities take care of the changes in the patterns and constraints.

Chapter 7- This chapter finally sums up the research work done with a critical analysis of contributions and milestones achieved discussions on future direction of work.

Chapter 2

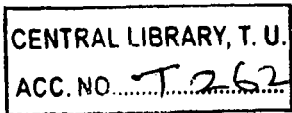
Review Work

2.1 Formal Concept Analysis

2.1.1 Theoretical Background

Formalization of human thinking helps in fostering the process of learning by giving a well-defined representation to human thoughts. Formal Concept Analysis (FCA) finds its core here. It considers a *concept* as a formal unit of human thought. FCA takes an input table specifying a set of objects and the properties there of, and finds all the *natural* clusters of properties and all the *natural* clusters of objects in the input data, where

- a *natural* object cluster is the set of all objects that share a common subset of properties, and
- a *natural* property cluster is the set of all properties shared by one of the natural object clusters. Natural property clusters corresponds



one-to-one with natural object clusters.

The input table is termed as the *formal context*. A pair of natural object cluster and natural property cluster forms a *concept*. This is based on the philosophical understanding that a *concept* is constituted by two parts: its *extension* which consists of all objects belonging to the concept, and its *intention* which comprises of all attributes shared by those objects. These concepts exhibit a subsumption hierarchy when organized into a lattice. A set of algorithms based on tools, like implications, exploration etc. is used for construction of such a formal concept lattice. Various operations like scaling, pruning etc. and visualization tools like the various line diagrams help in maintaining such a formal concept lattice of a given formal context. These operations facilitates further inferences from conceptual human thoughts explicitly represented as formal concepts. Pioneering works of Ganter and Wille [4, 1] lays the foundation of the formal framework for FCA. Basic definitions from these works which would be further used in this research are given here.

Definition 1 A **formal context** K is defined as a three tuple $\langle O, A, I \rangle$ which consists of two sets O and A and a relation I between O and A . The elements of O are called the objects, the elements of A are called the attributes of the context and the relation I indicates which objects have which attributes.

Definition 2 A **formal concept** is defined as a 2-tuple $\langle O_i, A_i \rangle$ derived from a formal context where

- $O_i \subseteq O$, O is a finite set of objects in the formal context.
- $A_i \subseteq A$, A is a finite set of attributes in the formal context.

- Every object in O_i has every attribute in A_i
- For every object in O that is not in O_i , there is an attribute in A_i which that object does not have
- For every attribute in A that is not in A_i , there is an object in O_i that does not have that attribute

O_i is called the extent and A_i the intent of the concept

The extent and intent of a formal concept follows a duality principle similar to Galois connection

Definition 3 (A, \leq) and (B, \leq) are two posets. A **Galois Connection** between these two posets consist of two monotone functions $F: A \rightarrow B$ and $G: B \rightarrow A$ such that $\forall a \in A$ and $\forall b \in B$ we have $F(a) \leq b$ iff $a \leq G(b)$

The set of formal concepts in a formal context exhibits an order by the subconcept-superconcept relationship among them. This order is defined as a concept lattice [1]

Definition 4 Let (O_1, A_1) and (O_2, A_2) be formal concepts in a formal context $\langle O, A, I \rangle$. The set of all concepts of $\langle O, A, I \rangle$ partially ordered by the subconcept relationship \leq is called the **Formal Concept Lattice** where \leq forms a Galois Connection as follows - $(O_1, A_1) \leq (O_2, A_2) \Leftrightarrow O_1 \subseteq O_2 \Leftrightarrow A_2 \subseteq A_1$

FCA tries to capture and visualize conceptual constructs in a context in such a manner that given the representation of a concept, we could derive

all other concepts that may have contributed to it. Thus, we could find out the relationships among the concepts in the form of *inheritance*. Organizing concepts as an inheritance hierarchy in the form of a concept lattice has been the approach of FCA. We illustrate by the common example of integer digits and their types. The concept lattice for this context as given in Figure 2.1¹ can be built either with respect to the *MEET* operation or *JOIN* operation on the context -

MEET - Here we start at the concepts having one attribute in the intentions. *MEET* operation on each pair of concepts results in a concept where the intention has all the attributes from both the concepts and the extension has the common objects between them. Carrying this forward we would reach at a set of concepts where no more *MEET* operation can be performed. At this point the concepts are closed.

JOIN - Here we start at the concepts having one object in the extensions. *JOIN* operation on each pair of concepts results in a concept where the extension has all the objects from both the concepts and the intention has the common attributes between them. Carrying this forward we would reach at a set of concepts where no more *JOIN* operation can be performed. At this point the concepts are closed.

¹Figure courtesy Wikipedia

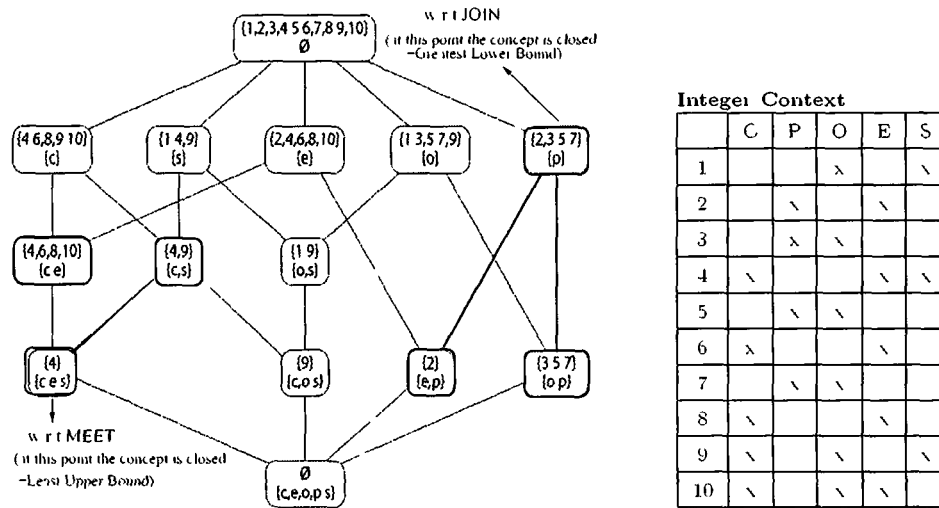


Figure 2.1 Integer Concept lattice

Definition 5 An **Inheritance hierarchy** is a rooted directed graph $G(V, E, \leq)$

where

r = Root node

\leq = Some lexicographic order

V = A set of nodes ordered with respect to \leq

$E = \{(a, b) \mid \text{for some } V_1 \subseteq V \text{ and } V_2 \subseteq V \text{ such that } V_1 \cap V_2 = \emptyset, a \in V_1, b \in V_2 \text{ and } a \leq b\}$

2.1.2 Research and Usage of FCA

Research and usage of FCA has been in different areas since the pioneering work done during the early 1980s. Initially it started to be used in the

area of Knowledge Representation and Reasoning (KR&R). Applications of FCA in the area of KR&R has been chiefly in ontology, Natural Language Processing (NLP) and linguistics [5, 6, 7, 8, 9, 10, 11]. FCA is used to enhance an ontology by converting it into a formal concept lattice and navigate in it for learning purpose. In NLP, knowledge descriptions are stored in a lexical. However ambiguity remains between the text actually used and the entries into the lexical database. To remove this, a detailed lexicon is required along with an ontology. On the other hand, for a detailed lexicon, a corresponding ontology is a requirement. So, as a starting point to the process of building the detailed lexicon, FCA techniques are used to bootstrap it with an initial concept lattice. Subsequent exploration techniques discover further specific entries. In addition to this FCA has been used in conjugation with Ripple Down Rule (RDR) to detect relationships among rules for knowledge reuse [12]. This initial application of FCA in the area of KR&R has been followed by applications in other areas like Software Engineering (SE), Data Mining (DM) and Concept Classification (CC). In the area of DM, FCA has been used chiefly for the purpose of extracting a hierarchy of mined information from voluminous data [13, 14, 15, 16, 17]. FCA usage has been found to be highest in the area of SE where it is chiefly used for the purpose of finding related artifacts from existing code [18, 19, 20, 21, 22, 23, 24, 25, 26]. CC has been used in conjugation with Knowledge Discovery in Databases (KDD) to classify concepts extracted from unstructured databases [27, 5, 17, 28, 12, 29, 30, 31]. Also, FCA has been used for ontology engineering to merge different ontologies into a concept lattice and classify concepts from it which is otherwise not possible in the source ontologies [32, 33, 34].

FCA is still researched and used actively through individual efforts and conferences like the International Conference on Formal Concept Analysis (ICFCA). It is found to be most prolific in the area of CC. In the area of CC, [35] is a work that gives an approach for clustering results returned by a search engine. [36] gives an approach of retrieving selective clusters from the input contexts by constraining concepts as per user requirements, while [37] gives an approach of refining results returned by web search engines. [38] gives an approach of mapping a Conceptual Graph to FCA while [39] is a work that deals with ontology completion. In the area of KR&R, [40] extends the navigation and annotation features of a standard search interface by creating a conceptual neighborhood of formal concepts built out of the search results and allowing users to navigate in this neighborhood.

Research contribution and usage of FCA over a period of 15 years across four areas viz. SE, DM, KR&R and CC are shown in Figure 2.2. Out of the 55 works reviewed, a total of 11 works have been in the area of KR&R, 27 have been in the area of SE, 6 have been in the area of DM and 20 have been in the area of CC. Thus, the focus of FCA has been the highest in the area of SE and very low in the area of DM. Though less in focus, FCA usage in KR&R has continued till the late 2000s. The high focus of FCA in SE occurred during the period of early 1990s to the late 2000s. The focus in DM has been minimal and concentrated during the first half of 2000s. FCA has been used in CC from the later part of the 1990s. Though it is difficult to project a definite usage and research trend of FCA from these figures, a shift from KR&R to SE can be seen. It could be indicative of FCA usage in the areas of Artificial Intelligence (AI) and Machine Learning (ML). This could

be substantiated from the fact that FCA has been used in SE chiefly for the purpose of finding related artifacts from legacy codes thus automating the process of restructuring software. These reviewed papers, along with their focus areas, contribution and tools developed if any are tabulated in Appendix A

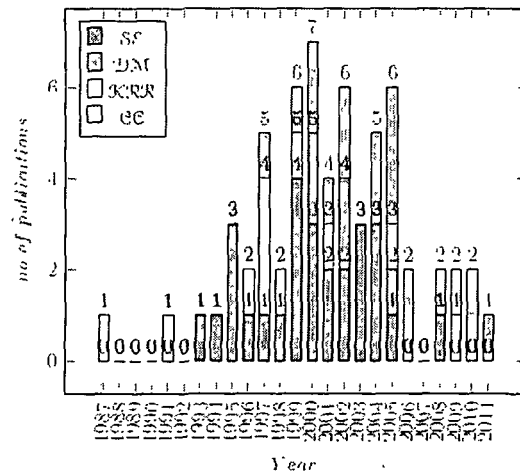


Figure 2.2: Usage of FCA

2.1.3 Existing Concept Lattice.Generating Algorithms

Algorithms that generate the concepts in a given context and builds the lattice deals with two problems, firstly how to generate all concepts and secondly how to avoid repetitive generation of the same concept. For avoiding repetitive generation of same concepts, an algorithm runs a specific canonicity test. Optimality of the algorithm depends on the time taken for generation of all the concepts. It is known that the number of concepts can be exponential

in the size of the input context in some cases. Hence, a concept generating algorithm can be considered optimum if it generates the concepts in the concept lattice with *polynomial time delay* and takes linear storage space for the generated concepts. An algorithm is said to have *polynomial time delay* if it executes at most polynomial steps of computations before generating the next concepts or halting. Each of these algorithms differ in how they exit the loop, how the closures are computed and how the canonicity test for avoiding repetitive generation of concepts is done. These algorithms can be compared and classified according to a set of seven properties [41]. We choose Ganter's Next-Closure for the purpose of generation and navigation of pattern lattices in our approach.

Ganter's Next-Closure Algorithm

Ganter's NextClosure [42, 41, 43] generates the concepts according to a lexicographic order of the subsets of attribute set A or object set O in a given formal context $\langle O, A, I \rangle$. Set Set_1 is considered to be lexicographically less than Set_2 if $min((Set_1 \cup Set_2) - (Set_1 \cap Set_2)) \in Set_2$. In general, a subset is considered to be lexicographically less than its superset. We denote the lexicographic order by \prec . Each step of the algorithm has two passes

- Generation of a new concept from the current set of attributes A_{cur} - The extent of the new concept is generated by intersecting the extents of all the concepts having the attributes in A_{cur} . A forms the intent.
- Canonicity test - The closure O''_{cur} of the current extent O_{cur} is calculated. If O''/O doesn't contain any object $q \prec max(O_{cur})$, then the

generation of the new concept is considered canonical. Here $\text{max}(O_{cu})$ gives the maximal element in O according to \prec . If the test fails, the algorithm continues with the next attribute set in the lexicographic order. If the test succeeds, the algorithm continues with next set of attributes obtained from the closure of A as follows - Include the maximal element not in the closure A''_{cu} and remove all elements from A''_{cu} that are lexicographically bigger than the newly included element. Thus the next set is constructed as $A''_{cu} \cup \{g\} \setminus \{h|h \in A''_{cu} \& g \prec h\}$ where $g = \text{max}\{O/A''_{cu}\}$

The algorithm continues until the canonical cover of A_{cu} is equal to A . Thus it is seen that Next-Closure doesn't require an auxiliary storage since it doesn't use already generated concepts to generate newer ones. This serves as an advantage of Next-Closure. Again use of lexicographic order considers selected subsets of the attribute set for generation of newer concepts, thus reducing the time required. The *polynomial time delay* of Next-Closure is $O(|A^2||O||L|)$, where L is the number of concepts generated.

2.2 Workflow and Workflow Pattern

Workflow continues to evolve over the years as a standard technology of the process domain. Research and development in workflow have chiefly concentrated in composition of a workflow from existing components to model a process at hand. [44-45, 46-47] deals with composition of workflow in the area of business process, [44-48, 49-51, 46-47, 52] deals with workflow composition chiefly from business processes in the form of web services.

[53, 50, 54, 55] deals with issues regarding various scientific workflows and their composition [56] is the only work found amongst the reviewed works that deals with a generic workflow. Thus the focus of generic workflow is found to be least amongst the research community [57] is a work that deals with dynamic structural changes at task level in a Workflow Management System (WFMS) based on a conceptual graph based model. Workflow technology is found to have evolved chiefly in two directions - business processes workflow and scientific application workflow as is revealed by the plot of the reviewed papers in Figure 2.3. Appendix B gives a detailed tabulation of these reviewed papers along with tools, deliverable developed and their respective areas of focus and contributions.

Contemporary workflow systems, either business process or scientific application addresses workflow requirements as different features which have varied suitability, perspectives and granularity. However, such features exhibit patterns inherently as is true to any other system and exhibit of these patterns in the workflow are constrained by the relationships amongst them. Research and development in the areas of workflow pattern and pattern-based workflow have been focused in identification and specification of workflow Patterns in various domains like bioinformatics [58], BPR [59, 60], SOA [61] Knowledge-flows [62, 63] BPM [64, 65, 66, 67] generic workflow patterns [65, 68, 67], Access-control [69]. Also, focus of research on WP is found in various application and programming platforms like Oracle [70], Java [71], Windows [72], LOTOS [73]. In this effort, different repositories of WPs have been developed. Explorations of contemporary workflow systems by Van Der Aalsts' group in this regard has established a robust reposi-

tory of WPs [74, 75, 76, 77, 78]. This repository has been accepted largely across industry and academia as a conceptual basis for process technology [64, 73, 79, 70, 80, 72, 71, 81, 82, 69]. However, an agreed upon framework for composition of a workflow from workflow patterns is found to be missing and this forms a part of this research. Plot in Figure 2.4 gives a area wise distribution of the number of papers reviewed while Appendix B tabulates in details the contribution, focus and tools developed by the research presented in these reviewed papers.

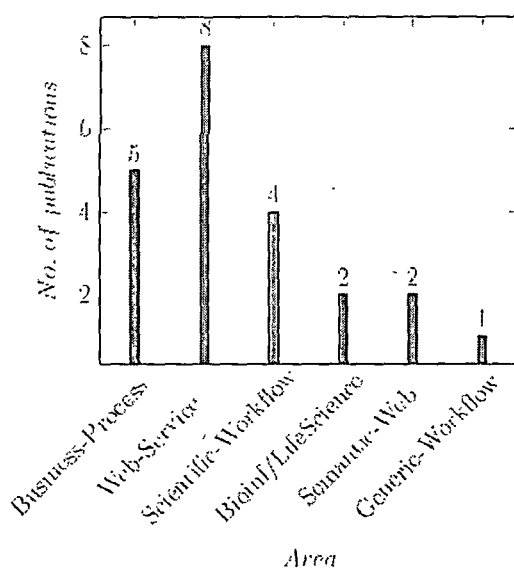


Figure 2.3: Area wise publications in workflow

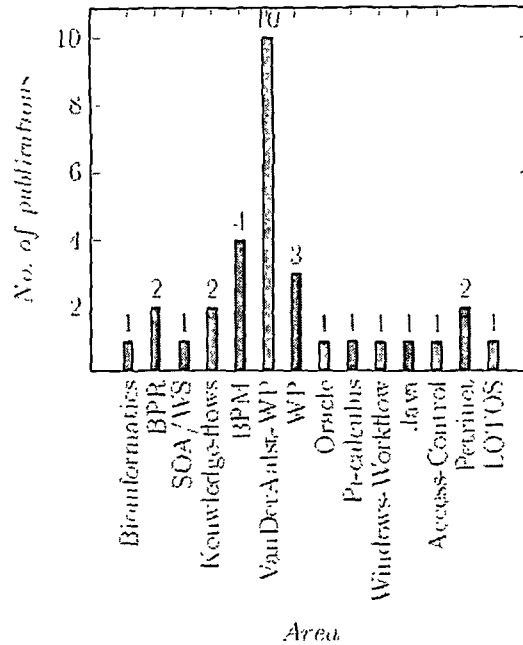


Figure 2.4: Area wise publication in WP

2.2.1 Van Der Aalst *et. al.* Repository of WP

With the primary aim to provide a conceptual basis for process technology, pioneering work in identification and documentation of workflow patterns were taken up jointly in Eindhoven University of Technology and Queensland University of Technology by Wil Van Der Aalst *et. al.* in 1999. These patterns have been aggregated and documented in the public domain as a well accepted repository of WPs across academia and industry. Here WPs have been captured in four different perspectives namely *Control-flow*, *Data*, *Resource*, *Exception Handling*. The *control-flow* perspective describes activities and their execution ordering through different constructors, which

permit flow of execution control e.g. sequence, splits, parallelism and join synchronization. The *data* perspective gives a layer of the processing data over the *control-flow* perspective. Documents and other objects which flow between activities and local variables of the workflow qualify in effect pre and post-conditions of activity execution. The *resource* perspective provides an organizational structure anchored to the workflow in the form of human and device roles responsible for executing activities. Typically, references to and workflow data are passed into and out of applications through activity-to-application interfaces, allowing manipulation of the data within applications. The control-flow perspective provides an essential insight into a workflow specification's effectiveness. The data flow perspective rests on it, while the organizational and operational perspectives are ancillary. These patterns have been documented with six attributes namely Description, Synonyms, Examples, Problem, Solution and Name. Examples state some instances of the pattern occurring in some process. Problem states a problematic situation that may arise in a process due to existence of the pattern. Solution discusses the approach taken by various Workflow Systems for handling the pattern. This descriptive way of documenting WPs helps us to have a good understanding of the design and architecture of Workflow Systems at large. However, it is of little help in maintenance of a Workflow in practice. In addition to this, the repository lacks a formal organization which could be the basis of finding the relationships amongst the patterns. A Workflow is an evolving system where changes need to be accommodated from time to time. These changes can be formalized as WPs and integrated into the Workflow.

2.3 Security and Security Pattern

Security solutions of a system has been largely seen from a pattern-based perspective as is evident from the large number of related works available. A Security Pattern (SP) captures expert knowledge of a security solution for a security problem within a defined context. SPs are structural prescriptions to the application developer to incorporate security solutions in a system. Methodologies for elicitation of security requirements has been the chief area of study for security patterns [83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98]. Apart from this, security patterns have been studied through different phases of software development lifecycle [99, 100], anti-patterns[101, 102] etc.

2.3.1 Repository of Security Patterns

A large corpus of SP repositories is being developed which is expanding rapidly - [98] gives a hierarchy of patterns for security of agent based system, [103] gives a classification of systems security patterns, [104] gives a catalog of architectural patterns. Based on Open Group template for pattern, [105] classifies security patterns for structural design, A comprehensive catalog of surveyed SPs based on security concern dimensions [Washizaki et al (2009)] and quality characteristics [Laverdiere et al (2006)] is given in [106], [107] Gives a new classification of SPs based on the Gamma et al's characteristics - Creational, Structural, Behavioral and Viega's and McGraw's 10 principle [108] gives a list of 14 SPs classified according to the CIA model [109] gives a list of AAPs categorized into pattern classes, [110] Gives a classification of

SPs based on multiple dimensions where each dimension cuts across a continuous concern space giving multiple regions [111] proposes seven patterns for application security [112] proposes 26 SPs for web application development [113] enhances security of sendmail by a set of SPs. Since the contexts of these repositories vary largely, the SPs have context-specific descriptions and classifications which differ from one another. However, all these repositories are seen to follow the general security properties under consideration for a secured system like authenticity, authority, integrity, confidentiality, non-repudiation, availability, separation of duty, binding of duty etc. An SP may address more than one property while there may be more than one SP that addresses the same property. Moreover, the patterns may be ordered from general to more specific classes based on different parameters like platform, pre-conditions etc. Due to the granularity of knowledge captured in design and deployment, SPs from different repositories are observably different. For example, SERENITY SPs in different levels of SERENITY framework, like organization, workflow and service, network and device levels are coherent but differ observably from those cited above. Schumacher gives a two dimensional organization of SPs [114] where one dimension considers three levels of abstraction viz. enterprise, system and application while the other dimension considers three phases of life-cycle - architecture, design and operation [115-108] an organization based on taxonomy and linguistic metaphors while [116-117, 118-108-119] attempt enumerations based on the Common Criteria for Information Technology security Evaluation. In addition though in-depth, these studies have been made from analysis of security scenarios of existing systems. Hence there is an absence of a formal organization

amongst these classification which could properly capture and represent the underlying security concerns in a platform independent manner. Most of these classifications are tag based and tangled with implementation details.

2.4 Allen's Interval Algebra

Allen's Interval Algebra (IA) deals with the problem of representing temporal knowledge. It introduces an interval-based temporal logic together with a computationally effective constraint propagation reasoning algorithm. In order to control the amount of deduction performed automatically by the system, a notion of *Reference Interval* is introduced.

Definition 6 *A Reference Interval RI is a cluster of time intervals (i_1, i_2, \dots, i_n) such that*

- *Each interval in the cluster is related to this reference interval by one or more of the 13 Allen's temporal relationships*
- *For each ordered pair from the cluster, the temporal relationships between the intervals in the pair is completely calculated*

Intervals can be represented by modeling their end points. Assuming a model consisting of a fully ordered set of points of time, an interval is an ordered pair of points with the first point less than the second. A key fact used in testing whether some condition P holds during an interval T is that if a smaller interval t is during T and P holds during t, then P also holds during T. This *during* relationship can be used to define a hierarchy of intervals in

which propositions can be “inherited”. Furthermore, such a *during* hierarchy allows reasoning processes to be localized, so that irrelevant facts are never considered. Allen proposes 13 relationships that could hold between any two intervals X and Y (Table 2.1). To facilitate multiple relationships holding between a pair of intervals, Allen also provides a network representation. Here a node representing an interval X is denoted by N_X . Allen computes the transitivity relations between *Reference Intervals* as in Table 2.2. Based on the transitivity table, Allen’s framework further provides a procedure for calculating transitive constraints between two intervals. The procedure as given in Algorithm 1 is used further on in this thesis for composition and change accommodation in a workflow.

Table 2.1: Allen’s 13 interval relations

Relation	Symbol	Illustration	Symbol of Inverse	Relation on end-points	Network representation
X before Y	<	xxx yyy	>	$X+ < Y-$	$N_X -- (<) \rightarrow N_Y$
X equals Y	=	xxx yyy	=	$(X- = Y-) \text{ and } (Y+ = X+)$	$N_X -- (=) \rightarrow N_Y$
X meets Y	m	xxxxyy	mi	$X+ = Y-$	$N_X -- (m) \rightarrow N_Y$
X overlaps Y	o	xxxx yyyyy	oi	$(X- < Y-) \text{ and } (X+ > Y-) \text{ and } (X+ < Y+)$	$N_X -- (o) \rightarrow N_Y$
X during Y	d	xxx yyyyyyy	di	$(X- > Y-) \text{ and } (X+ < Y+)$	$N_X -- (d) \rightarrow N_Y$
X starts Y	s	xxx yyyyy	si	$X- = Y-$	$N_X -- (s) \rightarrow N_Y$
X finishes Y	f	xxx yyyyy	fi	$X+ = Y+$	$N_X -- (f) \rightarrow N_Y$

Table 2.2: Transitivity table for the 12 temporal relations in Allen's IA omitting '='

Br_1C	$\langle \rangle$	d	d_i	o	o_i	m	m_i	s	s_i	f	f_i
Ar_1B	\langle	no info	\langle, o, m	\langle	\langle, o, m	\langle	\langle, o, m	\langle, o, m	\langle	\langle, o, m	\langle, o, m
	\rangle	no info	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i
d	\langle	\langle	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m
	\rangle	\rangle	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i	\rangle, o_i
d_i	\langle, o, m	o_i, o, d, d_i	o, d, d_i	o, d, d_i	o, d, d_i	o, d, d_i	o, d, d_i	o, d, d_i	o, d, d_i	o, d, d_i	o, d, d_i
	\rangle, o_i	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$	$m_i, d_i, s, =$
o	\langle	\rangle, o_i, o, d, s	\langle, o, m	\langle, o, m	s, o, o_i	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m
	\rangle	d_i, m_i	d_i, f_i	d_i, f_i	s_i, d, d_i	s_i, d, d_i	s_i, d, d_i	s_i, d, d_i	s_i, d, d_i	s_i, d, d_i	s_i, d, d_i
o_i	\langle, o, m	o_i, d, f	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s	\rangle, o_i, o, o_i, s
	\rangle	d_i, f_i	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$	$m_i, d_i, s_i, d, d_i, m_i$
m	\langle	\rangle, o_i, o, d, s	\langle	\langle	o, d, s	\langle	\langle	\langle	\langle	\langle	\langle
	\rangle	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i
m_i	\langle, o, m	o_i, d, f	\rangle, o_i, d, f	\rangle, o_i, d, f	$s, s_i, =$	\rangle, o_i, d, f	\rangle, o_i, d, f	\rangle, o_i, d, f	\rangle, o_i, d, f	\rangle, o_i, d, f	\rangle, o_i, d, f
	\rangle	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i
s	\langle	\rangle	\langle, o, m	\langle, o, m	o_i, d, f	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m	\langle, o, m
	\rangle	d	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i
s_i	\langle, o, m	o_i, d, f	d_i	o, d, f_i, o_i	o, d, f_i, o_i	o, d, f_i, o_i	o, d, f_i, o_i	o, d, f_i, o_i	o, d, f_i, o_i	o, d, f_i, o_i	o, d, f_i, o_i
	\rangle	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i	d_i, f_i
f	\langle	\rangle	d	$\rangle, o_i, \langle, o, d$	\rangle, o_i, m	\rangle, o_i, m	\rangle, o_i, m	\rangle, o_i, m	\rangle, o_i, m	\rangle, o_i, m	\rangle, o_i, m
	\rangle	d	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i
f_i	\langle	$\rangle, o_i, \langle, o, d$	d_i	o	o_i, d_i, s_i, m	o_i, d_i, s_i, m	o_i, d_i, s_i, m	o_i, d_i, s_i, m	o_i, d_i, s_i, m	o_i, d_i, s_i, m	o_i, d_i, s_i, m
	\rangle	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i	m_i, d_i

Algorithm 1 Calculate Transitive Constraint

CalTransCons(C1, C2) {Given any three intervals T1, T2 and T3. C1 and C2 are the constraints between T1,T2 and T2,T3 respectively}

We are to calculate C3, the constraint between T1 and T3. Let ε be the null constraints, r_a be a temporal relation in C1 and r_b be a temporal relation in C2

$C3 \leftarrow \varepsilon$

for all r_a in C1 **do**

for all r_b in C2 **do**

$C3 \leftarrow C3 \cup T(r_a, r_b)$ { r_a and r_b are $T(r_a, r_b)$ is the corresponding entry in Table 2.2 }

Return C3

2.5 Temporal Role Based Access Control

Role Based Access Control (RBAC) [120–121] is a widely studied Access Control scheme where permissions for executing a task are grouped together to roles and these roles are assigned to users for some sessions. A user could be a human or an autonomous agent. Temporal RBAC (TRBAC) [2] incorporates the following temporal dimensions into RBAC:

1. Periodic role enabling and disabling
2. Support for temporal dependencies among such actions. These dependencies are expressed by means of role triggers. Role triggers are active

rules that are automatically executed when the specified action occurs

3 Individual exception

4 Support for the first and third dimension on request basis during run-time

This TRBAC model, based on the RBAC model consists of the following components

- Sets $Users$, $Roles$, $Permissions$, and $Sessions$, representing the set of users, roles, permissions, and sessions, respectively,
- $PA: Roles \rightarrow Permissions$ the permission assignment function, that assigns to roles the permissions needed to complete their jobs,
- $UA: Users \rightarrow Roles$ the user assignment function that assigns users to roles,
- $user: Sessions \rightarrow Users$, that assigns each session to a single user,
- $role: Sessions \rightarrow 2^{Roles}$, that assigns each session to a set of roles, and
- $RH \subseteq Roles \times Roles$ a partially ordered role hierarchy (written \geq)

For enabling and disabling of roles over defined periods, the Role Enabling Base (REB) construct is defined in TRBAC

Definition 7 A Role Enabling Base (REB) is a set of elements of the following kinds

- 1 *Periodic events of the form $(I P p E)$ where I is a time interval P is a periodic expression $p E$ is a prioritized event expression with $p \prec T$*
- 2 *Role triggers of the form $E_1, \dots, E_n, C_1, \dots, C_k \rightarrow p E$ after Δt E_i 's are simple event expressions C_i 's are role status expressions $p E$ is a prioritized event with $p \prec T$ Δt is a duration expression*

For representing periodicity of roles, a *Periodic Expression* has been defined based on the notion of calendars

Definition 8 *A Calendar is defined as a countable set of contiguous intervals numbered by integers called indexes of the intervals. Given two calendars C_1 and C_2 we say that C_2 is a sub calendar of C_1 ($C_2 \sqsubseteq C_1$) if each interval of C_2 is exactly covered by a finite number of intervals of C_1*

Definition 9 *Given calendars C_d, C_1, \dots, C_n a **Periodic Expression** P is defined as $P = \sum_{i=1}^n O_i C_i \triangleright \iota C_d$, where $O_1 = \text{all}$, $O_i \in 2^I \cup \{\text{all}\}$, $C_i \sqsubseteq C_{i-1}$ for $i=2, \dots, n$, $C_d \sqsubseteq C_n$ and $\iota \in I$*

The symbol \triangleright separates the first part of the periodic expression identifying the set of starting points of the intervals it represents from the specification of the duration of each interval in terms of calendar C_d

Syntax of the constructs used in these definitions are given below

Priorities: Let (Prios, \preceq) be a totally ordered set of priorities. We assume that Prios contains at least two distinct numbers \top and \perp such that for all $v \in \text{Prios}$, $\perp \preceq v \preceq \top$

Simple event expressions: These have the form $enableR$ or $disableR$ where R is a role attached to the REB

Prioritized event expressions. These have the form $p \ E$, where $p \in P_{\text{prior}}$ and E is an event expression

Role Status expressions: These have the form $enabledR$ or $\neg enabledR$ where R is a role attached to the REB

Conflicting events Let R be an event. If enabling R gives rise to a conflicting situation, then the subsequent action of disabling R to overcome the conflict is expressed as $conf(enableR) \stackrel{def}{=} disableR$. Similarly $conf(disableR) \stackrel{def}{=} enableR$

The review work was primarily done keeping in mind resolution of the issues involved. FCA was reviewed in search of a formal organizing tool for patterns. Thus Formal Concept Lattice from FCA framework has been used for constructing formal organization of WPs and SPs. Review work on workflow and workflow patterns was done to explore for repositories of WPs and some architecture of constructing and maintaining a pattern-based workflow. Van Der Aalst *et al*'s repository was found to be the most comprehensive one and widely accepted. However a formal framework of these patterns was found to be missing, based on which an architecture for composition of a workflow in terms these patterns and then constraints could be achieved. The same forms a part of this research. Temporal constraints amongst WPs have been considered for composing workflow. In this regard Allen's IA was reviewed for ways and formalisms of representing a workflow and its patterns.

in terms of time intervals and relationships amongst them. Having realized that a workflow could be viewed as a hierarchy of sub processes where each sub process is a time interval during execution, each WP is represented as a *Reference Interval*. In order to formulate access control of the workflow via roles, TRBAC was reviewed and the REB construct from the same was used for periodic enabling and disabling of roles attached with WPs. Review work on security was done for repositories of SPs and formal organization of the same. Though many repositories in varied contexts were found to be existing, a formal organization of the same was found to be missing and this forms a part of this research.

With these findings and directions from the review work, a conceptual architecture of the proposed approach for composing and maintaining a pattern-based workflow is given in the next chapter.

Chapter 3

An Architecture of the proposed approach

3.1 The Architecture

A workflow is an abstraction of real work. It is considered as a representation of a sequence of operations allotted to some roles. A role could be a human, an automation or some mechanism. Patterns are invariably exhibited in workflow, as like in any other system. However, the relationships amongst these patterns within a workflow is specific. It is the concern of the workflow composer to gather the knowledge of the workflow from user specification and give a consistent representation. Articulating this in terms of patterns and their relationships would make the task easier for the composer in two ways - firstly, considering workflow at pattern level allows to hide the complexity of the workflow, secondly, considering a change and the propagation of its effect

at pattern level for a workflow reduces the work to be done for the verification of the introduced change. With this understanding, the work reported in this dissertation is aimed at achieving an approach for composition of a workflow in terms of its constituent patterns and the temporal constraints amongst them. The approach facilitates a robust evolution of the workflow by incorporating security and changes over passage of time.

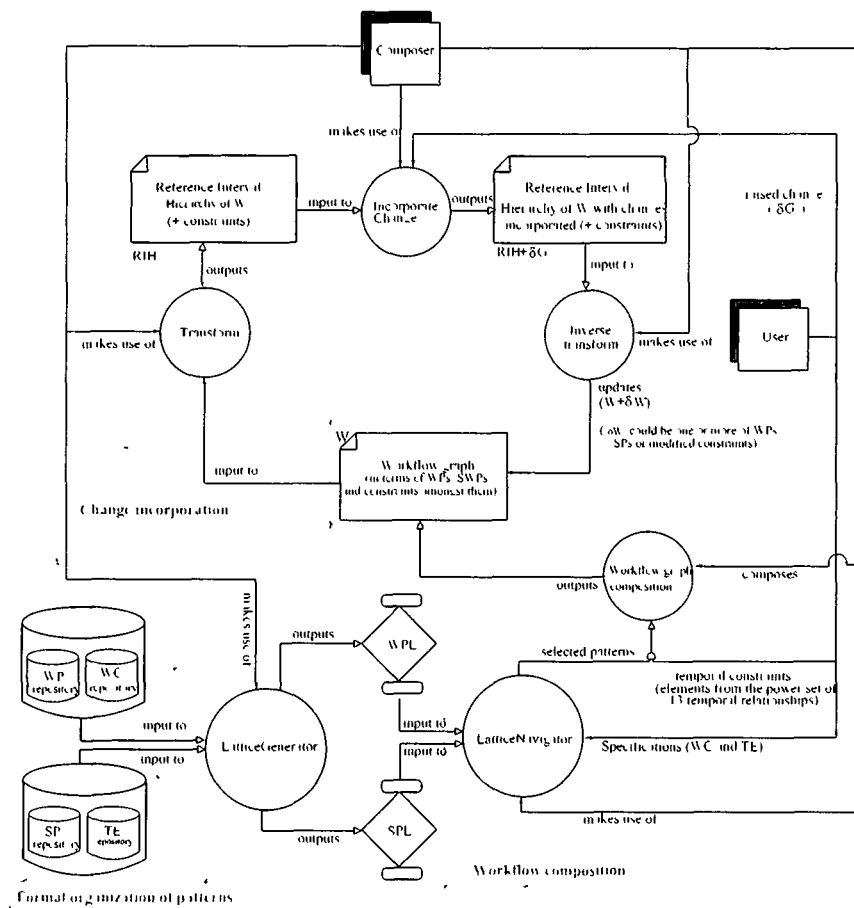


Figure 3.1 Architecture of the proposed approach

The overall architecture as depicted in Figure 3.1 involves three modules: (a) Formal organization of patterns; (b) Workflow composition and (c)

3.1.1 Formal Organization of Patterns as Lattice

In this module, depicted in Figure 3.2, a lattice theory based approach for formal organizations of patterns are done. This is in the form of pattern lattices of available WPs and SPs. WPs are characterized with Workflow Concerns (WC) such that WP and WC follow a Galois Connection. SPs are characterized by Trust Element (TE) such that SP and TE follow a *Galois Connection*. With these Galois Connections as basis, Workflow Pattern Lattice (WPL) is formed out of Van Der Aalst *et. al.*'s repository discussed in section 2.2.1. Similarly Security Pattern Lattice (SPL) is constructed out of the security requirements and concerns from the repositories reviewed in section 2.3.1. Generation and Navigation in WPL and SPL are done using the *LatticeGenerator* and *LatticeNavigator* algorithms. This is a one-time phase i.e. once the lattices are formed they can be used for composition of a workflow over and over again.

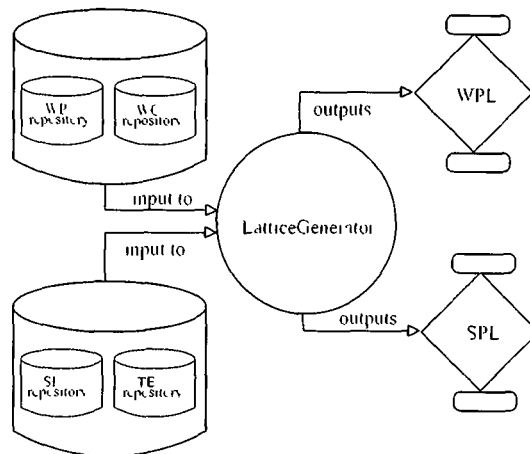


Figure 3.2 Pattern Lattice generation

3.1.2 Workflow Composition as Directed Graph

With WPL and SPL being generated, the workflow composition module is used to compose the workflow in the form of a directed graph of its patterns and the temporal constraints amongst them. Figure 3.3 shows the schematic of this module. User specifications of the workflow are formalized as WCs and TEs. With these WCs and TEs at hand, the composer makes use of the *LatticeGenerator* algorithm and selects the solution patterns from the WPL and SPL. These selected patterns along with the specified temporal constraints are input to the *ComposeWG* algorithm. This results in the composed workflow in the form of a directed graph where the patterns form the vertices and the temporal constraints form the edges. This is an iterative phase where the composer and user interact at each pass until the user satisfaction is achieved.

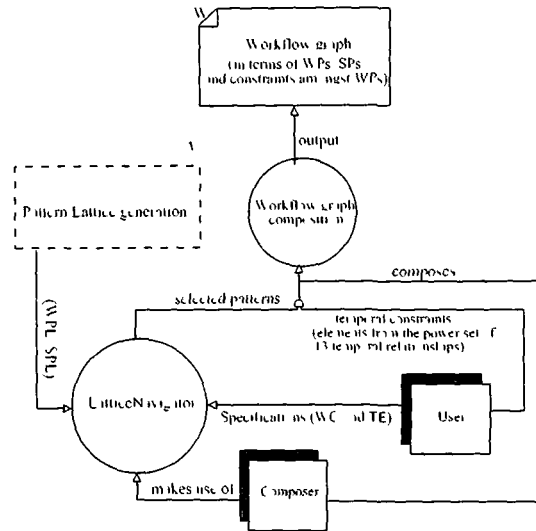


Figure 3.3 Workflow composition

3.1.3 Change Incorporation in a Workflow

Once the workflow is composed, this module as depicted in Figure 3.4 is used for performing the maintenance phase where changes raised by users during execution are incorporated into the workflow in a consistent manner. For incorporation of the changes, the composed workflow graph is input to the *Transform* algorithm. This traces a RIH from the graph. Thereafter changes in the form of new patterns being introduced, existing constraints being changed are introduced into the RIH by the *IntroduceNewPattern* and *ChangeConstraint* algorithms. Once the changes are incorporated, the updated RIH is transformed into its directed graph form by the *InverseTransform* algorithm. The workflow thus evolves and continues execution. This module is used over and over again in order to incorporate changes being raised

with passage of time.

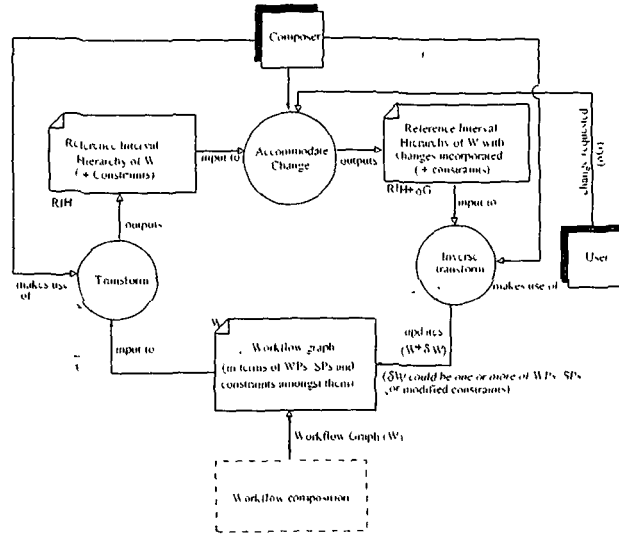


Figure 3.4: Change incorporation

3.2 Conceptual Layers of the Architecture

Conceptually, the architecture of the proposed approach involves WPs and SPs at three different layers with interfaces between them as depicted in Figure 3.5. Each interface has four elements that transform the lower layer to the upper layer - A transforming function, Input to the function which are the elements from the lower layer; Output from the function which forms the elements of the upper layer; Interacting entity that makes use of the function

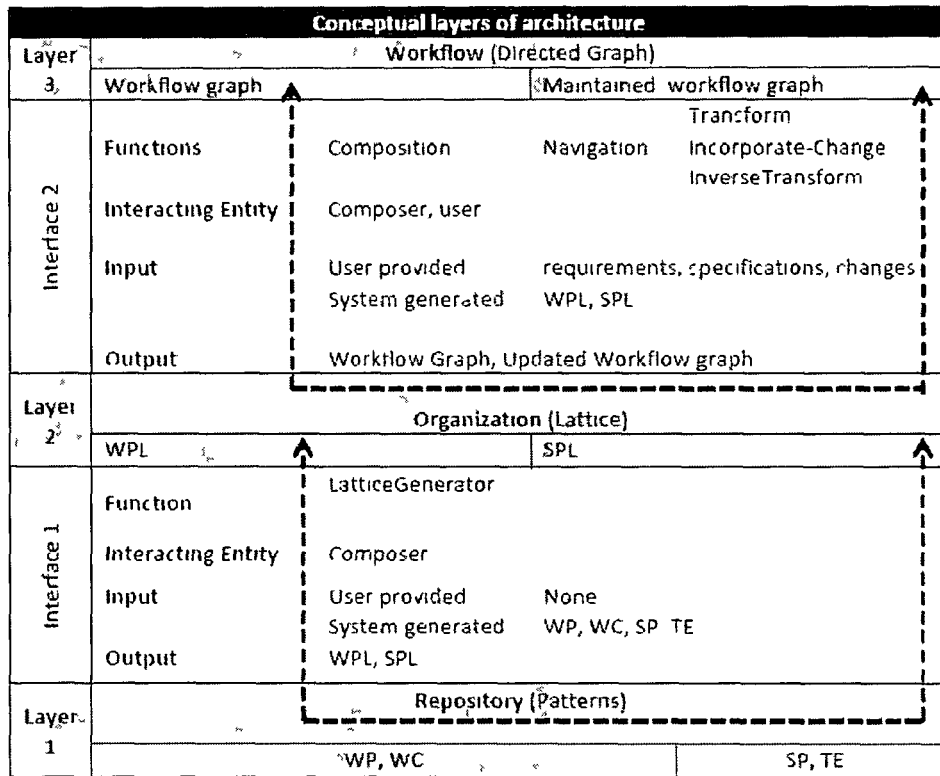


Figure 3.5 Conceptual layers

- **Atomic layer (Layer 1)** - Here the patterns exist independently. It consists of two repositories of patterns - Van Der Aalst *et al's* repository of WPs along with their characterizing WCs, SP repository along with their characterizing TEs.
- **Organizational layer (Layer 2)** - Here the patterns from layer 1 are organized into formal concept lattices. The WP repository from layer 1 is organized into a WPL and the SP repository is organized into a SPL. The Interface 1 that transforms layer 1 to layer 2 has the following

elements -

Function This is Ganter's Next-Closure based lattice generation procedure that generates the pattern lattices from the pattern repositories in layer 1 by preserving the sub-concept and super-concept relationships amongst them. This function is separately used for WPs and SPs to generate the WPL and SPL respectively.

Interacting entity This is the composer who maintains the repositories of the WPs and SPs and makes use of the lattice generation procedure for generating the pattern lattices.

Input The WPs and WCs from layer 1 form the input to the pattern lattice generating procedure for WPL. Similarly, the SPs and TEs form the input for SPL. There are no user specified inputs in this interface. Therefore, this interface is interacted only by the composer and is used in the pre composition phase of the architecture.

Output The WPL and SPL form the output in this interface.

- **Workflow layer (Layer 3)** - Here the patterns selected from the WPL and SPL based on user specifications are composed into a workflow with the constraints amongst them. The Interface 2 that transform layer 2 to layer 3 has the following elements -

Function This interface consist of five functions - a workflow composition function that composes the workflow as a directed graph of the selected patterns and the temporal constraints amongst them; a Ganter's next-Closure based navigation algorithm that

is used to navigate in the pattern lattices for selecting patterns based on user provided requirements, specifications and changes; a transform function that traces a RIH from a workflow graph; incorporate change function that adds a new pattern or changes an existing constraint in the RIH and an inverse transform function that transforms back the updated RIH back to it's workflow graph form.

Interacting entity The composer and user are the interacting entities in this interface. The composer makes use of the set of functions while the user interacts with the composer to provide the requirements, specifications and changes.

Input During the composition phase, the user provides the requirements and specifications to the composer. The composer in turn formalizes them to WCs and TEs and uses them as input to the navigation procedure for selecting the solution patterns. With these patterns and the temporal constraints amongst them provided by the user, the composer makes use of the composition function to compose the workflow. System generated inputs in this interface are the WPL and SPL from layer 2. This interface is used during the composition and maintenance phase of the architecture.

Output The workflow graph and the updated workflow graph after incorporation of changes form the output from this interface.

3.3 Enactment of the Architecture

Enactment of the architecture, depicted in Figure 3.3, involves three phases. In the first two phases, the elements of the interfaces amongst the conceptual layers described in Section 3.2 are spread across before and during workflow composition. In the third phase, the elements are spread across the maintenance of the workflow.

Stepwise breakdown of enactment of the architecture can be seen as a three-phased process:

- **Phase 1 (Prologue)** Enactment starts with this phase which consists of two steps - 'Build SPL' and 'Build WPL'. SP and TE repository is the input to 'Build SPL' while WP and WC repository is input to 'Build WPL'. The composer executes both these steps making use of the Pattern Lattice Generation procedure. Out from the two steps are the SL and WPL respectively. This phase is a one-time phase and is not required to be executed during the maintenance of the workflow.
- **Phase 2** This phase consists of four steps - 'Accept user specifications', 'Formalize user specification', 'Select solution patterns' and 'Compose Workflow Graph'. The first four steps are executed in iteration until solution patterns for all the user specifications are satisfactorily selected for subsequent composition of the workflow. The specifications communicated by the user in the first step is formalized by the composer in the second step into TE, WC and Temporal Constraints. With these formalized specifications, the composer executes the third step to select the solution patterns with the help of the pattern lattice navi-

Time		Composition				Maintenance			
		Input	Interacting entity	Function	Output	Input	Interacting entity	Function	Output
Phase 2	Prologue (Phase 1)	Building SPL	SP repository, TE repository	compose r	Pattern lattice generator	SPL (conceptual)			
		Building WPL	WP repository, WC repository	compose r	Pattern lattice generator	WPL (conceptual)			
	Iteration	Accept user specifications	User needs, requirements	User, compose r					
		Formalize user specifications	User needs, requirements	compose r		TE, WC, TC			
		Search solution patterns	TE, WC, SPL, WPL	compose r	Pattern lattice navigator	SP, WP			
	Output	Compose workflow graph W	WC, WP, SP	compose r	Workflow composition	W (conceptual) file (physical)			
	Iteration (Phase 3)	Transform to RIH				W	Composer	transform	RIH (conceptual) file (physical)
		Accept change				δG	user		
		Incorporate change in RIH				RIH, δG	composer	Constraint propagation	Updated RIH (conceptual) file (physical)
		Inverse Transform RIH				Updated RIH	composer	Inverse transform	Updated W (conceptual) file (physical)

Figure 3.6: Enactment of the architecture

gator. Once all the required solution patterns have been selected, the composer executes the fourth and last step in this phase to compose the workflow graph.

- **Phase 3** The third and final phase of enactment takes care of maintenance of the workflow after it has been composed in the second phase and execution starts. It consists of four steps - 'Transform to RIH', 'Accept Change', 'Incorporate Change in RIH' and 'Inverse Transform RIH'. These steps are executed whenever a change is raised by the user. The composer executes the first step to transform the workflow graph to a RIH. Thereafter the user raises the change and the composer executes the second step to accept this change for incorporation. With this the composer executes the third step to incorporate the raised change in a consistent manner into the RIH. Finally, the composer executes the fourth step to transform back the updated RIH to its original directed graph form and the workflow continues execution with this changes being incorporated.

In this chapter an architecture of the proposed approach for composing and maintaining a pattern-based workflow is given. It takes into account the inputs and interaction between the workflow composer and the user. The user comes up with requirements and the specifications for composition of a workflow and changes to be incorporated from time to time. In response the composer formalizes these requirements and specifications in the form of patterns selected from the pattern lattices and compose the workflow graph. In addition to this the conceptual layers and an enactment cycle of the architecture that gives the flow of execution of the steps involved in enactment of the proposed approach are given. With this architecture at hand the next four chapters deal with the details and the formalization of the modules involved in the architecture.

Chapter 4

Formal Organization of Patterns

Patterns are exhibited in a workflow like in any system. Structurally, these patterns would not be exhibited in isolation but would always be inter-related to one another. A pattern itself is a prescription to a recurring problem in a domain. Different people and different schools of thoughts working on a domain document these patterns separately. Absence of a formal structure of these patterns eludes them to fundamentally different semantics and remains distributed into different levels of expressiveness, even though they may lead to the solution of the same recurring problem. Also it is the concern of the workflow composer to gather the description of the system from the user's specifications and give a consistent representation. Articulating this description in terms of patterns and their relationships would make the task easier for the composer in two ways - firstly considering system at pattern level allows to hide the complexity of the system, secondly considering a

change and the propagation of its effect at pattern level for a system reduces the work to be done for the verification of the introduced change. Therefore, formal organization of patterns in a system could serve as a central point of reference from where the composer could draw instances for representation of the system in a structured and modular way.

In this chapter formal concept lattice from FCA is used to organize patterns. A formal concept lattice is a structure that organizes objects in a given context into a subsumption hierarchy based on a Galois Connection between the set of objects and the set of characterizing attributes of the object set. The organization would require generation of all the concepts in the given context and the subsumption hierarchy amongst them. Section 4.3.1 describes such an approach based on Ganter's Next-Closure algorithm. If one could characterize the set of patterns by a set of attributes which form a Galois Connection with the patterns, then formal concept lattice would serve as a feasible organization of patterns. With this basis WPs and SPs are organized into formal concept lattices here.

4.1 Formal Organization of Workflow Patterns

This approach to a formal organization of WPs is applicable to any pattern repository which could be characterized by a set of attributes that form a Galois Connection with the set of patterns in the repository. Here Van Der Aalst *et al*'s repository of WPs is considered for illustrating the approach.

4.1.1 Workflow Pattern Lattice

During the composition of a workflow, the composer has to understand the user specifications properly and identify the solutions in terms of patterns. Users communicate their specifications about the workflow in their own convenient way which is usually not structured enough to identify the solution pattern. It becomes the responsibility of the composer to capture these needs and specifications in a structured manner, identify the solution patterns and integrate them to the workflow. For this, we formalize Workflow Concern (WC). A Concern in a workflow is a localized proposition. It allows the user and the workflow composer to focus on local issues in the workflow. This localized approach to building a workflow is helpful to the composer from the point of modularity and maintenance. Again, pattern is a solution to a recurring problem in a domain. We consider user specifications formalized as Workflow Concern (WC) to be problems in the workflow domain to which WPs are the desired solutions. Problems and solutions follow a Galois Connection by nature, since larger the problem set at hand, smaller would be the solution set and vice-versa. Therefore WPs can be organized as concept lattice. This serves as the basis for building WPL using FCA techniques.

4.1.2 Basic Formalization

Definition 10 *A pattern P is a collection of tasks (with temporal constraints amongst the tasks) representing a recurring problem in a given context. Here P is characterized by $\langle T, TempConst \rangle$ where*

T - Set of constituent tasks of P

TempConst - Temporal Constraints amongst the tasks in T

A pattern needs to be considered in a certain context for real life application. The problem it exhibits may occur in more than one context but it's solution would be different for each of them. For example - Invalid reference occurs in both application and communication domain. The solution of Invalid reference in application domain is initialization while that in communication it is timeout and replay.

Definition 11 *A workflow is an abstraction of real work that operates under a system of forces and exhibits patterns with constraints between them. A workflow is characterized by a six tuple $\langle WF, S, F, P, C, R \rangle$ where*

WF = A pattern that represents the workflow in it's entirety.

S = A pattern it starts with

F = A set of one or more finish patterns. The workflow finishes with one of these patterns.

P = The set of patterns exhibited in the workflow, inclusive of the "start" pattern S and the patterns from the set of "finish" patterns F.

C = The set of constraints between pairs of elements from $P \cup WF$.

R = A set of roles (human or automated) that executes the patterns. R is a primitive for the workflow structure.

Definition 12 *A Workflow Pattern (WP) is defined as $\langle T, R_{wp}, \tau, TempConst, CRules, REB, V_{index} \rangle$ where*

- T is the set of tasks within the pattern
- $R_{sup} \subseteq R$ is the role set attached with the pattern, where R is the set of roles of the workflow in whose context this pattern is exhibited
- τ is a mapping from R to T assigning tasks in T to the roles in R
- $TempConst$ gives the Temporal constraints between pairs of elements of T where each constraint is one of Allen's Temporal Relationships
- $CFrules$ gives the set of rules for execution of an instance of this pattern. It enumerates all possible execution walks of the pattern when instantiated
- REB is the Role Enabling Base¹ attached with this WP
- V_{index} is the index of this WP in Van Der Aalsts' repository. This index is retained for easy referencing

Definition 13 A Workflow Concern (WC) is defined as a specification of a need in a workflow characterized as $\langle C, P, D \rangle$ where

$C(\text{Concern})$ Concern states the user's apprehension about some particular situation in a workflow which may be an issue to be resolved or an enhancement to be made. A concern is either a functional one viz coding, design/algorithm, data structure, initialization, approval/verification etc or a non functional one viz security, documentation, performance etc

¹See section 2.5 for Role Enabling Base

P(Perspective) This gives the workflow perspectives in which the concern is exhibited. It can have values from the set Data Resource Exception handling Control-flow Operational and is represented by a vector

D(Description) This states the user's apprehension in details. For the designer, this becomes a point of reference for selecting a WP providing a solution for the concern.

Definition 14 Workflow Context is a 3-tuple $\langle G_w, M_w, I_w \rangle$ where

$G_w = \text{Set of WPs}$

$M_w = \text{Set of WCs}$

$I_w = \text{Set of mapping from } G_w \text{ to } M_w$

For $g \in G_w$ and $m \in M_w$, $gI_w m$ relates WP g as a solution to WC m .

Definition 15 A Workflow Concept of the workflow context $\langle G_w, M_w, I_w \rangle$ is a pair (A, B) with $A \subseteq G_w$, $B \subseteq M_w$, $A' = B$ and $B' = A$ where

$$A' = \{m \in M_w \mid (\forall g \in A) gI_w m\}$$

$$B' = \{g \in G_w \mid (\forall m \in B) gI_w m\}$$

Here A' is the minimal set of WCs belonging to the WPs in A such that for every WP $g \notin A$ there is at least one WC in A' that does not belong to g . B' is defined accordingly. $\mathcal{W}(G_w, M_w, I_w)$ denotes the set of concepts in the context $\langle G_w, M_w, I_w \rangle$.

Definition 16 :Workflow Pattern Lattice Let (A_1, B_1) and (A_2, B_2) are workflow concepts in $\mathcal{W}(G_w, M_w, I_w)$. The subconcept relationship denoted by \leq is defined as follows

$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 \Leftrightarrow B_2 \subseteq B_1$. The set of all workflow concepts of $\langle G_w, M_w, I_w \rangle$ partially ordered by this relation and denoted by $\bar{\mathcal{W}}(G_w, M_w, I_w)$ is called the *Workflow Pattern Lattice*.

4.2 Formal Organization of Security Patterns

A large and expanding corpus of SP repositories is being developed as is revealed from the review work. The SPs have context-specific descriptions and tag-based classifications which differ from one another. However, all these repositories are seen to follow the same set of general security properties under consideration for a secured system. In addition, though in-depth, these repositories have been made from analysis of security scenarios of existing systems. Hence there is an absence of a formal organization amongst these classification which could properly capture and represent the underlying security concerns in a platform independent manner. This absence of formal structure of the SP repositories is resolved in this research by organizing them into a formal concept lattice. Security here is characterized by *Trust* instead of the usual *Threat* based proposition. Trust by nature is a user-centric proposition in any security context. When considered, it makes the system more user friendly and increases the answerability of the user. Figure 4.1 presents the schematic of the trust-based model taken in this approach.

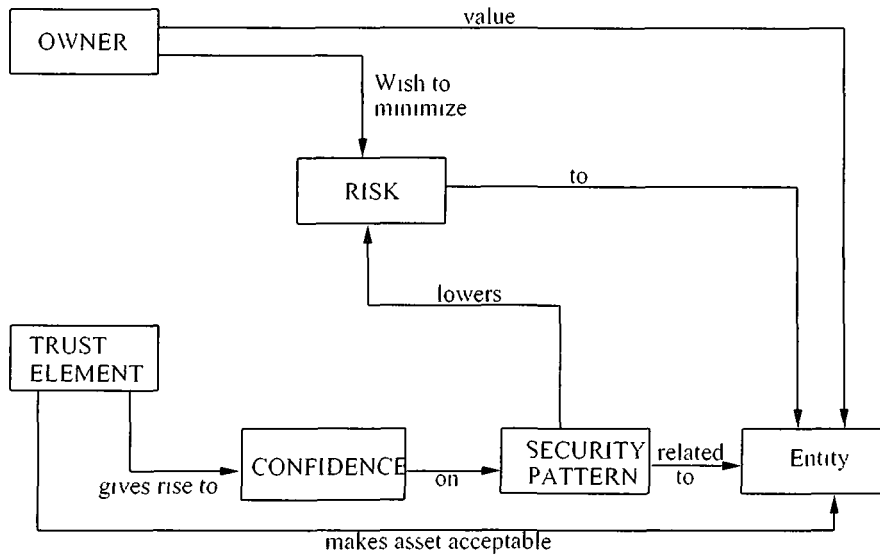


Figure 4.1 Trust-Based Security Model

4.2.1 Security Pattern Lattice

In order to be organized into a concept lattice SPs needs to be characterized by some attribute such that they follow a Galois Connection. The trust-based approach to security taken here serves this purpose. 'security' and 'trust' follows a Galois Connection by nature since larger the trust, smaller would be the security requirement and vice-versa. Here we formalize trust as 'Trust Element' (TE) and characterize SP as a task to be performed for enforcing some TEs. Therefore SPs can also be organized as a concept lattice. This serves as the basis for generating a SPL using FCA techniques.

4.2.2 Basic Formalization

Definition 17 : Trust Element (TE) *is a property or a statement about an entity in a context which is otherwise unknown and whose absence makes the entity vulnerable to certain attacks. Here an entity could be a subject, a object or a situation in a given context. It is characterized as $\langle P, C, T, E \rangle$ where*

P(Property) - *The property defining the element.*

C(Confidence) - *Confidence is the current degree of trust that is achieved by the TE after spending a period of time in the context in conjunction with it's client entity. Application of SP(s) keeps on increasing the Confidence on the trust element.*

T(Threshold) - *Threshold is that measure of Confidence which makes the TE acceptable. Once achieved, the entity to which the TE is bound is considered to be fully secured.*

E(Entity) - *The entity to which the TE is bound. It is a contextual measure. In a workflow security context, it can have values from { Data, Resource, Control flow Operational, Presentation}*

Definition 18 Security Pattern(SP) *is a task that needs to be executed for enforcing a Trust Element in the security context of a domain. It is characterized by $\langle Precondition, Task, Exhibit, PostCondition \rangle$ where*

Precondition(Optional) - *Conditions that needs to be satisfied before this SP could be executed. This could be other SPs executed or some contextual conditions being satisfied*

Postcondition(Optional) - *Conditions that needs to be satisfied after this SP is executed This could be other SPs executed or some contextual conditions being satisfied*

Task - *Describes the task that needs to be executed*

Exhibit - *The pattern demonstrates itself by Exhibit if executed*

This definition of SP is induced from Common Criteria for Information Technology Security Evaluation [122] which is accepted as the common minimum criteria for selection of SPs

Definition 19 Security Context is a 3-tuple $\langle G_s, M_s, I_s \rangle$ where

$G_s =$ Set of SPs

$M_s =$ Set of TEs

$I_s =$ Set of mapping from G_s to M_s

For $g \in G_s$ and $m \in M_s$ $gI_s m$ tells us SP g is to be applied in absence of TE m

Example of mapping I_s could be $gI_s m$ where g is Authenticator SP and m is Authority TE

Definition 20 4 Security Concept of the security context $\langle G_s, M_s, I_s \rangle$

is a pair (A, B) with $A \subseteq G_s$, $B \subseteq M_s$, $A' = B$ and $B' = A$ where

$$A' = \{m \in M_s | (\forall g \in A) gI_s m\}$$

$$B' = \{g \in G_s | (\forall m \in B) gI_s m\}$$

$S(G_s, M_s, I_s)$ denotes the set of concepts in the context $\langle G_s, M_s, I_s \rangle$

Definition 21 Security Pattern Lattice Let (A_1, B_1) and (A_2, B_2) are security concepts in $\mathcal{S}(G_s, M_s, I_s)$. The subconcept relationship denoted by \leq is defined as follows

$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 \Leftrightarrow B_2 \subseteq B_1$. The set of all security concepts of $\langle G_s, M_s, I_s \rangle$ partially ordered by this relation and denoted by $\tilde{\mathcal{S}}(G_s, M_s, I_s)$ is called the Security Pattern Lattice

4.3 Generating Concept Lattice

Ganter's Next-Closure generates all the concepts from a given formal context. However, the procedure doesn't preserve the information about the subsumption hierarchy that exists amongst the generated concepts. In a subsumption hierarchy, any concept inherits from its immediate superconcepts or Least Upper Bounds (LUB). This information is crucial as it could be exploited for faster navigation through the lattice. A lattice is generated by preserving the subconcepts and superconcepts of each generated concept, i.e., by preserving the subsumption hierarchy of the concepts. The single attribute concepts arranged in Ganter's lexicographic order is provided as initial input to the procedure. With this, the *LatticeGenerator* procedure is devised for generating the lattice from the context at hand.

4.3.1 Concept Lattice Generating Procedure

Inputs:

Single Attribute Concepts - Concepts having single attributes in the m-

tents *MaxConcept* - Maximal concept having all objects in the extent
MinConcept - Minimal concept having all attributes in the intent

Outputs:

ConceptLattice - ConceptLattice generated

Variables:

CurIntent - Current intent calculated, *CurExtent* - Current extent calculated
NewConcept - Current concept generated *NewConcepts* - generated concepts from current iteration
PrevNewConcepts - new concepts from previous iteration, *CurConcepts* - new concepts from previous iteration

Step 1: Append *MaxConcept* as a superconcept of each concept in *SingleAttributeConcepts* and make them subconcepts of *MaxConcept*

Step 2. Initialize *PrevNewConcepts* to *SingleAttributeConcepts*

Step 3: Repeat Steps 3.1 to 3.5 until $PrevNewConcepts = \emptyset$ or *CurConcept* = *MinConcept*

Step 3.1: For each subset $S_{P \setminus C}$ of *PrevNewConcepts* starting at the first subset in the lexicographic order generate *NewConcept* as follows

- $CurIntent = \text{Union of the intents of all the concepts in } S_{P \setminus C}$
- $CurExtent = \text{Intersection of the extents of all concepts in } S_{P \setminus C}$

- $NewConcept = (CuiIntent \ CuiExtent)$

Step 3.2: check for canonicity of $NewConcept$ as follows

- Calculate $CuiExtent'' \setminus CuiExtent$ the set of objects in closure of $CuiExtent$ but not in $CuiExtent$
- $NewConcept$ is canonical and the test succeeds if $CuiExtent'' \setminus CuiExtent$ doesn't have an element lexicographically less than the maximal element of $CuiExtent$

Step 3.3: If the canonicity test in Step 3.2 succeeds then the procedure continues with the next subset of concepts calculated from $CuiIntent$ as follows

- Include the maximal element not in $CuiExtent''$ that are lexicographically bigger than the included element. Let this set be denoted by $TempIntent$
- Remove all subsets in $PrevNewConcepts$ having concepts with intent which are lesser than $TempIntent$. This is because the concepts that would have been generated by the removed sets would have already been generated

The procedure continues with the next subset in $PrevNewConcepts$ if the test fails

Step 3.4: If $NewConcept$ is canonical and is not the minimal concept then append it to $CuiConcepts$ and $ConceptLattice$ make all the concepts in S_{PNC} as the superconcepts of $NewConcept$. Append $NewConcept$ as a subconcept of each concept in S_{PNC}

Step 3.5: If all the subsets in *PrevNewConcepts* have been exhausted then initialize *PrevNewConcepts* with *New Concepts* and clear *New Concepts*

Once the procedure terminates *ConceptLattice* would have the lattice in it with the superconcepts and subconcepts marked for each of the concept

Algorithm 2 LatticeGenerator (*SingleAttributeConcepts*, *MaxConcepts*, *MinConcepts*)

Input: *SingleAttributeConcepts* - Initial set of concepts having single attribute
in extent *MaxConcept* - Maximal concept *MinConcept* - Minimal concept

Output: *ConceptLattice* - The concept lattice generated

```

1  for each concept  $C_{S_{AC}} \in \text{SingleAttributeConcepts}$  do
2    Append ( $\text{MaxConcept}$ ,  $\text{SuperConcepts}(C_{S_{AC}})$ ) Append ( $C_{S_{AC}}$ ,
      SubConcepts( $\text{MaxConcept}$ ))
3   $\text{SingleAttributeConcepts} \leftarrow \text{PrevNewConcepts}$ 
4  repeat
5    for each subset  $S_{P\wedge C} \in \text{PrevNewConcepts}$  do
6       $\text{CurIntent} \leftarrow 0$   $\text{CurExtent} \leftarrow 0$ 
7      for each concept  $C \in S_{P\wedge C}$  do
8         $\text{CurExtent} \leftarrow \text{CurExtent} \cup \text{Extent}(C)$ 
9      for each concept  $C \in S_{P\vee C}$  do
10      $\text{CurIntent} \leftarrow \text{CurIntent} \cup \text{Intent}(C)$   $\text{CurExtent} \leftarrow \text{CurExtent} \cap$ 
       $\text{Extent}(C)$ 
11      $\text{NewConcept} \leftarrow (\text{CurIntent}, \text{CurExtent})$   $\text{TempIntent} \leftarrow 0$ 
12     if  $\text{Min}(\text{CurExtent}' \setminus \text{CurExtent}) > \text{Max}(\text{CurExtent})$  then
13       for each  $a \in \text{CurIntent}$  do
14          $\text{Set}_{\geq a} \leftarrow$  Set of elements in  $\text{CurIntent}$  lexicographically greater than
           $a$ 
15          $\text{TempIntent} \leftarrow \text{TempIntent} \cup \text{Max}(\text{Set}_{\geq a} \setminus \text{CurExtent}'')$ 
16       for each subset  $S \in \text{PrevNewConcepts}$  do
17         if  $\exists \text{Con} \in S$  &&  $\text{Intent}(\text{Con}) \leq \text{TempIntent}$  then
18           Remove( $S$   $\text{PrevNewConcept}$ )
19         if  $\text{NewConcept} <> \text{MinConcept}$  then
20           Append( $\text{NewConcept}$   $\text{CurConcepts}$ ) Append( $\text{NewConcept}$ 
       $\text{NewConcepts}$ ) Append( $\text{NewConcept}$   $\text{ConceptLattice}$ )
21         for each  $\text{CON} \in S_{P\vee C}$  do
22           Append( $\text{CON}$   $\text{SuperConcept}(\text{NewConcept})$ ) Append( $\text{NewConcept}$ 
      SubConcept( $\text{CON}$ ))
23         if  $\text{PrevNewConcepts} = \emptyset$  then
24            $\text{PrevNewConcepts} \leftarrow \text{NewConcepts}$   $\text{NewConcepts} \leftarrow \emptyset$ 
25 until  $\text{PrevNewConcepts} = \emptyset$  ||  $\text{CurConcept} = \text{MinConcept}$ 

```

The polynomial time delay of Ganter's algorithm is $O(|M|^2|G||L|)$, where L is the number of concepts generated [42, 41, 43], G is the set of object and M is the set of attributes in the context. This is the time required to produce a concept. In the lattice generating procedure described here the only extra computational step involved is the marking of superconcept and subconcept of a new concept. When a new concept is generated it is made the subconcept of each concept in the current subset and each concept in the current subset is made its superconcept. Thus the maximum number of computation steps for this would be $|L| + |L| = 2|L| = O(|L|)$. Hence the polynomial time delay of the procedure remains same as that of Ganter's algorithm - $O(|M|^2|G||L|) + O(|L|) = O(|M|^2|G||L|)$. The pseudocode for this *LatticeGenerator* is given in Algorithm 2.

4.3.2 Generating the Workflow Pattern Lattice

A stepwise approach based on lattice theory and FCA and as depicted in Figure 4.2 is taken for generating the WPL.

Rule set for WP Enumeration- The following set of rules is used for enumerating WPs and WCs in a workflow context. These rules are in conformation with the requirements for building a concept lattice from a set of concepts [123].

- **Complete with respect to the workflow context.** Each WP traces to at least one WC.
- **No spurious WCs.** Each WC has at least one WP related to it.

- **Must follow a Galois Connection:** The set of WCs and the set of WPs follow a *Galois connection* which states if we increase the number of WCs in a workflow concept, the number of WPs decreases and vice versa
- **Multi-valued WCs:** Each WC may be multi valued in nature where its values differ in the ‘perspective’ attribute

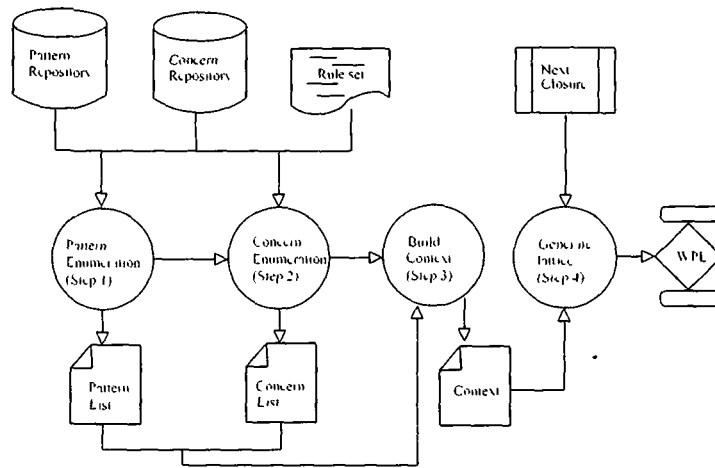


Figure 4.2 Flow of steps in generation of WPL

Step 1 Workflow Patterns Enumeration

Van Der Aalst et al.'s repository discussed in Section 2.2.1 is used for enumerating the WPs. Any new pattern that may arise in the repository can be accommodated if it follows the rule set considered here. Appendix D gives a partial listing of the WPs enumerated.

Step 2 Workflow Concerns Enumeration

There are no formal rules for extracting workflow “concerns” from the context at hand. The composer and user’s tacit understanding of the contextual issues and their needs to enumerate WCs is considered in this regard. Though analogous in essence, WC should not be confused with “concern in software engineering” [124, 125, 126] where it is used as a criteria for modularizing evolving softwares.

Step 3 Constructing Workflow Context

The context is build as a cross table depicted in Table 4.1 where WCs are tabulated along columns and WPs along rows. The cell at the cross section of a WC and it’s related WP is marked. These WCs and WPs are enumerated in Appendix D.

Table 4.1 Workflow Pattern Context

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
WP1	x	x											
WP2			x	x			x	x	x				
WP3			x	x					x				
WP4			x			x	x		x				
WP5			x	x		x	x	x	x				
WP6			x	x					x				
WP7			x	x	x	x	x						
WP8	x			x			x		x			x	
WP9	x						x					x	
WP10							x	x	x				
WP11	x						x						
WP12	x						x						
WP13												x	x
WP14										x	x	x	x

Step 4 Classifying Workflow Concepts

Making use of the *LatticeGenerator* algorithm, the WPL is generated from the context in Table 4.1. The WPL, as visualized in Figure 4.3, classifies WPs as formal concepts $\{WPs, WCs\}$. The supremum of the lattice is the concept $\{\{all\ WPs\}, \phi\}$ while the infimum is the concept $\{\phi, \{all\ WCs\}\}$. As we move down from the supremum, WCs are introduced gradually into the concepts. With an increase in the WCs, there is a decrease in the WPs in a concept. This leads us to the observation that concepts are more general towards top and tends to get specific towards the bottom of the lattice.

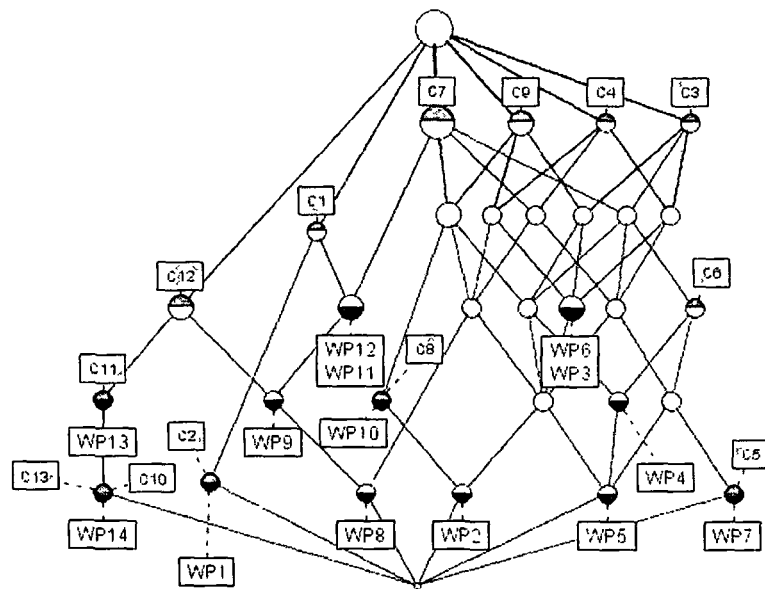


Figure 4.3: Workflow Pattern Lattice

4.3.3 Generating the Security Pattern Lattice

The SPL is generated similarly as WPL was generated. The set of SPs induced from the SP corpus discussed in Section 2.3.1 and as per the rule set used for WPL forms the extent

	ID	AP	RT	ATH	MBS	INT	CON	PRV	FRM	CMP	CNF	CNS	PRS	UNQ
ATH	\	\		\		\	\	\	\		\	\		
ATR	\	\		\		\	\	\	\		\	\		\
CP	\		\			\							\	
DID	\	\		\		\	\	\			\			\
UDE	\										\			
RB	\		\			\							\	
PF				\		\				\	\	\		
SPF					\	\			\				\	
ECON						\	\						\	
SYN	\			\	\		\					\	\	
DMA					\	\								
RFR					\	\								
IND	\	\						\						
LG											\			
NRM						\					\	\		
Acronyms used														
ATH	AUTHENTICATOR						ID	IDENTITY						
ATR	AUTHORIZER						AP	ACCESS POLICY						
CP	CHECKPOINT						RT	RETENTION						
DID	DEFENSE IN DEPTH						ATH	AUTHORITY						
UDE	USER DEFINED EXCEPTION						MBS	MEMORY BUFFER SIZE						
RB	ROLLBACK						INT	INTEGRITY						
PF	PROOF						CON	CONFIDENTIALITY						
SPF	SECURE.PREFORK						PRV	PRIVILEGE						
ECON	ESTABLISH.CONNECTION						FRM	FORMALITY						
SYN	SYNCHRONIZATION						CMP	COMPLETENESS						
DMA	DYNAMIC.MEMORY.ALLOCATOR						CNF	CONFIRMITY						
RFR	REFRESH						CNS	CONSISTENCY						
IND	INDUCTION						PRS	PERSISTENCE						

LG	LOGIC	UNQ	UNIQUENESS
NRM	NORMALIZATION		

Table 4.2: Security Context

There are no formal rules as such for extracting TEs from literature and documentation of the context in hand. The user's tacit understanding of trust and complementing documented threats in some cases is used for listing the TEs in Appendix E and this forms the intent. Figure 4.4 visualizes the SPL for the security context in Table 4.2.

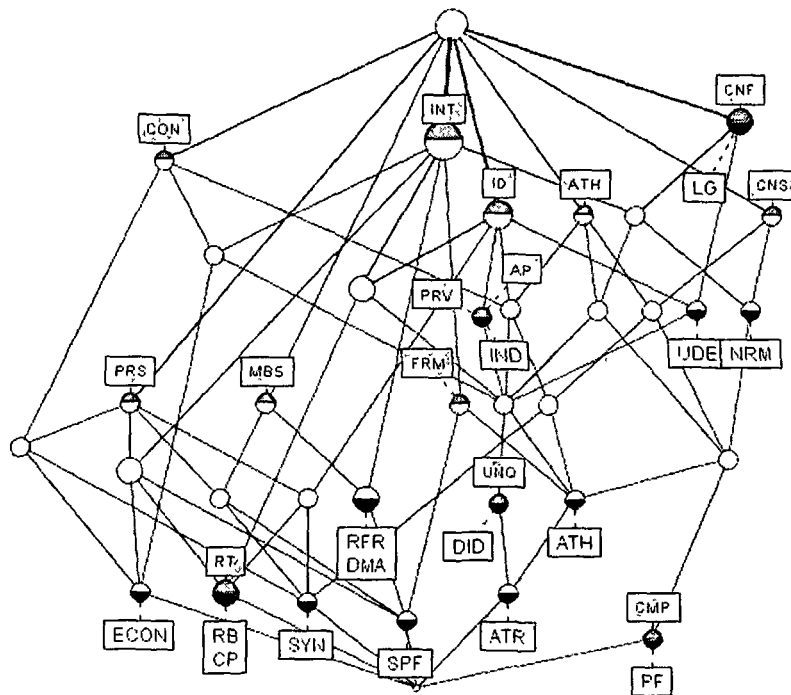


Figure 4.4: Security Pattern Lattice

4.4 Navigating in Pattern Lattices

WPL and SPL categorizes patterns to concepts arranged in a subsumption hierarchy. In the lattice, subconcepts and superconcepts of a concept are preserved along with it. Therefore, navigation in the lattice could be efficiently done by browsing the subconcept or superconcept hierarchy starting from the minimal or maximal concept.

4.4.1 Scope of Navigation

A lattice may be navigated for different purposes - to find a particular concept, to find a particular set of concepts, to find a particular object in the extent of a particular concept etc. In the scope of this research, it is required to navigate for a set of patterns that serve as a solution to a given set of attributes. For WPs, the set of attributes would be the user-chosen WCs and for SPs, the set of attributes would be the user-chosen TEs. Hence the scope of navigation is to find a concept having the user-chosen set of WCs or TEs as a subset of its intent. The extent of this concept will give the required set of patterns.

4.4.2 Navigation Criteria

The user-provided set of WCs/TEs form the navigation criteria. With this set, the navigation starts at the minimal concept having the complete set of WCs/TEs as the intent and stops at a concept whose intent is lexicographically more than the provided set.

4.4.3 Navigation Procedure

Inputs

ConceptLattice - List containing the concepts in the lattice in the lexicographical order. Each entry has a concept and pointers to its subconcepts and superconcepts. *Required Attributes* - Set of attributes for which the solution patterns are to be found. For WPL it will have WCs and for SPL it will have TEs.

Variables

Cur Concept - The current concept being explored. Initialize it to the first concept in *ConceptLattice*. *Cur SubConcepts* - Queue containing concepts yet to be explored.

Step 1. Repeat the following steps until *ConceptLattice* is empty or concept desired is found. The desired concept will have an intent lexicographically greater than *Required Attributes*.

Step 1.1:

Add *Cur Concept* to *Cur SubConcepts*.

Step 1.2

Repeat the following steps until *Cur SubConcepts* have only *MinConcept* or the desired concept is found.

Step 1.2.1.

If $\text{Intent}(\text{Cur Concept})$ greater than *Required Attributes*, then return *Cur Concept* as the desired concept. If not then append the subconcepts of *Cur Concept* to *Cur SubConcepts*.

and remove it from *Cur SubConcepts* and *ConceptLattice*

Step 1.2.2:

Make the $\text{Top}(\text{Cur SubConcepts})$ as the *Cur Concept*

Step 1.3.

Initialize *Cur Concept* as the next element in *ConceptLattice*. Remove the element from *ConceptLattice*

The pseudocode for this *LatticeNavigator* is given in Algorithm 3

The objects in a context do not remain in isolation but works under a system of forces. The forces define how the objects interact and conflict with one another and also the goals to be achieved. As such, each object could have a number of related objects to it. In that case, a concept in the lattice would exhibit another lattice within it where the set of objects would be the union of the related objects of the objects in the parent concept and the set of attributes would be the union of the attributes of all the related objects in the parent concept. With this understanding at hand, we define a *Multi-Context* and a *Formal Multi-Concept*. The Next-Closure algorithm is extended to *MultiContextNavigator* in order to facilitate navigation in a Multi-Context.

Definition 22 Multi-Context is the representation of a (formal) concept A as a 3-tuple $\langle G_{mc}, A_{mc}, I_{mc} \rangle$ where G_{mc} is a set of objects corresponding to the related objects of all the concepts in A , A_{mc} is a set of attributes of all the related concepts in A and I_{mc} is a set of relations indicating which objects have which attributes

Algorithm 3 LatticeNavigator(*ConceptLattice*,

RequiredAttributes, *MaxConcept*, *MinConcept*)

[**LatticeNavigator**]

Input: *ConceptLattice* - The concept lattice to be navigated

RequiredAttributes - Set of attributes for which the solution pattern is to be found
MaxConcept - Maximal concept *MinConcept* - Minimal concept

Output: - *DesiredConcept* - Concept containing the solution pattern searched for

```
1 CuiConcept ← Top(ConceptLattice)
2 if CuiConcept <> NULL then
3   Append (CuiConcept CuiSubConcepts)
4   repeat
5     if Intent(CuiConcept) > RequiredAttributes then
6       DesiredConcept ← CuiConcept
7       RETURN DesiredConcept
8     else
9       Append(SubConcept(CuiConcept),CuiSubConcept)
10      Remove((CuiConcept) CuiSubConcept,ConceptLattice)
11      CuiConcept ← Top(ConceptLattice)
12      if CuiConcept == NULL then
13        RETURN NULL
14      until (MinConcept ∈ CuiSubConcepts && |CuiSubConcept| =
15        1) || Found(DesiredConcept)
15 else
16   RETURN NULL
```

Definition 23 A Formal Multi-Concept is a 3-tuple $\langle G_i, A_i, R_i \rangle$ derived from a Multi-Context $\langle G_{mc}, A_{mc}, I_{mc} \rangle$ where

- $G_i \subseteq G_{mc}$ and $A_i \subseteq A_{mc}$
- Every object in G_i has every attribute in A_i

- For every object in G_{mc} that is not in G_i , there is an attribute in A_i which that object does not have
- For every attribute in A_{mc} that is not in A_i , there is an object in G_i that does not have that attribute
- For every object in G_i , there is a subset of related objects in R_i

Algorithm 4 MultiContextNavigator(Rel_Obj, Vect_Selected_Obj, No_Atr, Atr)
[MultiContextNavigator]

Input: Rel_Obj - User provided related object Sel_Attrib - User provided attribute value for the object to be searched No_Atr - number of attributes in the context Atr - Array of attributes in the context

Output: Vect_Selected_Obj = A vector of the objects resulting from the search

```

1  Cui_Attrib_Indice ← 0
2  repeat
3    Find all subsets AllSubSet_CAI of Atr of size Cui_Attrib_Indice+1
4    Cui_Intent←0 Cui_Extent← MaxExtent Tai_Concept←0
5    for Each subset S ∈ AllSubSet_CAI do
6      Cui_Intent← S
7      Cui_Extent ← Intersection of the extents of Concepts from
          Sin_Attrib_Con corresponding to each element in S
8      if Cui_Extent <> 0 and Sel_Attrib ∈ Cui_Intent then
9        Tai_Concept={Cui_Intent Cui_Extent} Tai_Extent←Cui_Extent
10       for Each object Cui_Obj ∈ Tai_Extent do
11         if Cui_Obj is valid then
12           Search related objects of Cui_Obj for a match with Rel_Obj If a
              match found then add it to Vect_Selected_Obj
13         Break from current loop
14       Cui_Attrib_Indice ← Cui_Attrib_Indice+1
15  until Cui_Attrib_Indice = No_Atr

```

This algorithm for navigating in a Multi-Context can be used to search for a particular pattern in a concept in the SPL. Using the procedure, the user searches for some SP on the basis of a TE. This takes the user to a security concept in the SPL that has SPs satisfying the provided TE. For reaching at the desired SP within the concept at hand, the user launches a second level search on the basis of a precondition or in other words a related pattern. The related pattern submitted by the user is searched for in the precondition vector of each of the patterns in the extent of the security concept at hand. The search returns all those patterns whose precondition vector contains the related pattern submitted by the user in the second level search. Since the precondition of a SP is multivalued in nature, we define a multi-context security concept as a lattice viz. `MultiContextSC`.

Definition 24 A `MultiContextSC` is defined by a 3-tuple $\langle G_{ms}, A_{ms}, I_{ms} \rangle$ where

- G_{ms} is Set of security patterns in the concept
- A_{ms} is the union of all preconditions belonging to all patterns in G_{ms}
- I_{ms} is the set of relationship between elements of G_{ms} and A_{ms} that tells us which precondition is applicable to which pattern.

In this chapter, an approach for a formal organization of patterns has been achieved in the form of formal lattice. The two lattices viz. WPL and SPL will be used further on for composition of a pattern-based workflow. The WPL has been achieved from the Van Der Aalst's repository of WPs. Here WPs are characterized by WCs. The SPL has been generated from the

repository of SPs as discussed in Section 2.3.1. Here a trust-based approach has been taken and SPs has been characterized by TEs.

The *LatticeGenerator* procedure devised is applicable to a repository of patterns from any other context. This would require one to consider the repository of patterns as the extent and build a repository that would serve as the intent like the WCs for the WPL and TEs for the SPL.

The *LatticeNavigator* procedure checks if the intent of the currently visited concept in the lattice is lexicographically less than the set of required attributes provided. If so, this concept would provide the desired patterns and the procedure terminates. This navigating procedure is used in composition of the workflow graph as discussed in Chapter 5.

Chapter 5

Composition of a Workflow Graph

Constituent patterns in a workflow could follow different sequences of execution based on the constraints satisfied among them. Thus, a workflow could have a number of paths that may be followed for a particular execution. With this understanding and the formalization of workflow as given in Definition 11, Proposition 1 is established. This proposition serves as the foundation of the approach for composition of a workflow as a directed graph where the patterns form the vertices and the edges form the temporal constraints amongst them.

Definition 25 *A directed graph is an ordered pair $G(V,E)$ where*

$V = \{v_1, v_2, v_3, \dots, v_n\}$ is a finite set of elements called the vertices

$E = \{ (v_i, v_j) \mid 1 \leq i, j \leq n, (v_i, v_j) \text{ is an ordered pair} \}$ is the set of edges

Axiom 1 *Each pattern, exhibited within a workflow has a finite interval of execution during a successful execution of the workflow*

Proposition 1 *A workflow W is a directed graph G of its patterns and temporal constraints among them*

Proof

Let $W = (WF, s, F, P, \rho, R)$ be a workflow with temporal constraints such that

WF = Pattern that represents the workflow in its entirety

s = The 'start' pattern

F = The set of finish patterns

P = The set of constituent patterns including S and F

ρ = The set of temporal constraints between pairs of patterns in $P \cup WF$

Let C = Set of temporal constraints amongst elements of P $c_{ab} \in C$ be the temporal constraint between the patterns p_a and p_b in P . From Allen's IA framework any temporal constraint between two intervals will always have an inverse constraint between the intervals. This could be derived by considering the inverse of the constraint from Allen's 13 temporal relations. Therefore, for representing a workflow with n patterns considering half of the ordered pairs of patterns would suffice. This would be equal to combination of n with factor 2 given by ${}_n C_2 = \frac{n!}{2!(n-2)!}$

¹ ! is the symbol of factorial. $n!$ represents factorial of n .

Basis: $s=WF=f, f \in F, s = f, |F|=1, |P|=1$ (One pattern in W) In this case there could be two scenarios

- WF appears only once in W - In this scenario $\mathcal{V} = \{WF\}, \mathcal{C} = \{\phi\}$ Therefore $G=(\mathcal{V}, \mathcal{C})$ forms the null graph which is a directed graph with a single vertex
- WF appears more than once in W - Let $\mathcal{V} = \{WF, WF_t\}$ where $1 \leq t \leq h$ and WF_t is an executing instance of WF . Since these are instances of the same pattern, each instance could be temporally constrained with the rest instances by one of the temporal relations in $\{s \circ f, d, m, <\}$. Therefore $\mathcal{C} = \{c_i \mid 1 \leq i \leq \frac{(h+1)!}{2^{h(h-1)!}}, c_i = (s \circ, f_i, d, m, <)\}$. Thus $G=(\mathcal{V}, \mathcal{C})$ forms a directed graph

Case 1: $|F|=1, f \in F, s \neq f, |P|=2, s \notin F$ (Two patterns in W) In this case there could be the following scenarios

- s and f executes once each - Here $\mathcal{V} = \{s, f\}$. From Definition 11, a workflow has a single start pattern and an execution of the workflow would end in one of the finish patterns. Hence, the only possible temporal constraints between s and f are $\{o, m, f\}$. Therefore $\mathcal{C} = \{(o, m, f)\}$ and thus $G=(\mathcal{V}, \mathcal{C})$ forms a directed graph
- s and f executes more than once each - Here $\mathcal{V} = \{s, s_j, f, f_k\} \mid 1 \leq j \leq n, 1 \leq k \leq d, s_j$ is an executing instance of start pattern s and f_k is an executing instance of finish pattern f . The only possible constraints between an instance of start pattern s and an instance

of finish pattern f are $\{o_i, m, f\}$. Similarly the temporal constraints between any two start pattern instances and any two finish pattern instances are $\{< m \ d \ f_i\}$. Therefore $C = \{c_i\}$, $1 \leq i \leq \frac{(n+d+2)!}{2^{!(n+d+1)}}$. Thus $G=(\mathcal{V}, C)$ forms a directed graph.

- s executes once and f executes more than once - Similarly as for the previous cases $\mathcal{V} = \{s, f, f_k\}$, $1 \leq k \leq d$, $C = \{c_i\}$, $1 \leq i \leq \frac{(d+2)!}{2^{!(d+1)}}$. Thus $G=(\mathcal{V}, C)$ forms a directed graph.
- s executes more than once and f executes once - Similarly as for the previous cases $\mathcal{V} = \{s, s_j, f\}$, $1 \leq j \leq n$, $C = \{c_i\}$, $1 \leq i \leq \frac{(n+2)!}{2^{!(n+1)}}$. Thus $G=(\mathcal{V}, C)$ forms a directed graph.

Case 2: $s \neq f$, $|P| = 3$, $p \in P$ (Three patterns in W) Here the proof is given for multiple execution of s , p and f . Other scenarios would follow similarly.

$\mathcal{V} = \{s, s_j, p, p_l, f, f_k\}$, $1 \leq j \leq n, 1 \leq l \leq e, 1 \leq k \leq d$. Possible temporal constraints are as follows:

- Constraint between start pattern instances - $\{<, m \ d \ f_i\}$
- Constraint between finish pattern instances - $\{< m \ d \ f_i\}$
- Constraint between start pattern and finish pattern instances - $\{o_i, m, f\}$
- Constraint between start pattern and intermediate pattern instances - $\{< o_i, m, f, d\}$
- Constraint between intermediate pattern and finish pattern instances - $\{<, m, o_i, s, d\}$

Therefore $C = \{c_i\}, 1 \leq i \leq \frac{(n+e+d+3)!}{2^{i(n+e+d+2)}}$. Thus $G = (\mathcal{V}, C)$ forms a directed graph

Case 3: Let the proposition hold for λ patterns $\lambda \geq 3$. Then the directed graph would have at most $\frac{\lambda!}{2^{i(\lambda-2)}}$ edges apart from the self-edges from and into each pattern

Let the pattern set be increased by one more pattern. In this case there could at most λ newer edges. Again, addition of an edge to an existing directed graph results in a directed graph in turn. Therefore, if the proposition holds for λ edges, then it also holds for $(\lambda+1)$ patterns.

Thus from Basis, Case 1, Case 2, and Case 3, the proposition is established by induction.

5.1 Composition without Security

Composition of the workflow is done as a directed graph of its patterns and temporal constraints between them. For introducing patterns and constraints to the workflow graph in a path-consistent manner, procedures *ConstraintAlongWalk* and *ValidateConstraint* are devised that assist in modularizing the composition procedure.

5.1.1 Constraint Along a Walk

Let $Cons$ be the relation that stores the constraints between the patterns in the workflow graph. i and j be the lexicographic indices of patterns A and

B. A consistent walk P between A and B would be of the form $P^{ij}_i, P^{ij}_{i+1}, \dots, P^{ij}_{i+k-1}, P^{ij}_{i+k}, P^{ij}_{i+k+1}, \dots, P^{ij}_{i+j-1}, P^{ij}_{i+j}$ where any two consecutive P^{ij} s are explicitly connected by a constraint. The transitivity constraint T along the walk P is calculated as:

$T = \text{Cons}[i,i+1]$, the entry in Cons that gives the temporal constraint between the vertexes P^{ij}_i and P^{ij}_{i+1}

$T = \text{CalTransCons}(T, \text{CalTransCons}(C_a, C_b))$. C_a and C_b are the temporal constraints between P^{ij}_i, P^{ij}_{i+1} and $P^{ij}_{i+1}, P^{ij}_{i+2}$ respectively. Here CalTransCons is Allen's procedure for calculating transitive constraint and is given in Algorithm 1

$T = \dots$

$T = \text{CalTransCons}(T, \text{CalTransCons}(C_l, C_m))$. C_l and C_m are the temporal constraints between $P^{ij}_{i-2}, P^{ij}_{i-1}$ and P^{ij}_{i-1}, P^{ij}_i

Algorithm 5 presents the pseudocode for this procedure.

5.1.2 Validating a Constraint

Let p_a and p_b be two patterns and $\text{Cons}[a,b]$ be the constraint between them that needs to be validated. Let m be the number of in-edges to p_a and n be the number of out-edges from p_b . Then there could be a total of $(m \times n)$ number of walks involving p_a and p_b . Each of these walks would start with an in-edge to p_a and end with an out-edge from p_b . Let W_{iabj} be such a walk with i^{th} in-edge to p_a as the start-edge and j^{th} out-edge of p_b as the

end-edge p_a and p_b are directly connected by $\text{Cons}[a, b]$ in W_{iabj} . Constraint along W_{iabj} would be validated for consistency if it is less than or equal to the constraints along all the existing walks involving p_a and p_b .

- Calculate T - the constraint along walk W_{iabj} using the *Constraint 4-longWalk* algorithm
- For each between p_i and p_j where p_a and p_b are not directly connected calculate CW_{ij} , the constraint along walk. If for each CW_{ij} , $T \subseteq CW_{ij}$, then $\text{Cons}[a, b]$ is validated for W_{iabj} else it is invalidated and the procedure terminates

$\text{Cons}[a, b]$ will be finally validated if it is validated for all the walks of type W_{iabj}

The pseudocode for this *ValidateConstraint* procedure is given in Algorithm 6

Algorithm 5 ConstraintAlongWalk(A,B,Cons)

Input. A, B - Patterns between which the constraint along walk is to be found
 Cons - Relation storing constraints between patterns in workflow graph

Output: T - The constraint along a walk P between A and B

- 1 $P_i^{ij} \leftarrow A, P_j^{ij} \leftarrow B$ { i and j are the lexicographic indexes of A and B in Cons }
- 2 $T \leftarrow \text{Cons}[i, i+1]$ {Constraint between the first two patterns in the walk P }
- 3 **for** $l=i+1$ to $j-2$ **do**
- 4 $T = \text{CalTransCons}(T, \text{CalTransCons}(\text{Cons}[l], \text{Cons}[l+1]))$

Algorithm 6 $\text{ValidateConstraint}(p_a, p_b, \text{Cons})$

Input: p_a, p_b - Patterns between which constraint is being validated
 Cons - Relation storing constraints between patterns in workflow graph
1 $m \leftarrow$ Number of in-edges to p_a $n \leftarrow$ Number of out-edges from p_b $T \leftarrow \phi$
 $\text{Flag} \leftarrow \text{VALID}$ {Flag keep tracks of the validity of constraint between p_a and p_b }
2 **for** $i = 1$ to m **do**
3 **for** $j=1$ to n **do**
4 $\text{IndirectWalks} = \text{FindIndirectWalks}(p_a, p_b)$ { FindIndirectWalks finds all the walks between p_i and p_j where p_a and p_b are not directly connected}
5 $T = \text{CalcTransCons}(\text{Cons}[i, a], \text{CalcTransCons}(\text{Cons}[a, b], \text{Cons}[b, j]))$,
6 **if** $\text{IndirectWalks} \neq \phi$ **then**
7 **for** each $w \in \text{IndirectWalks}$ **do**
8 **if** $T \subseteq \text{ConstraintAlongWalk}(w)$ **then**
9 CONTINUE
10 **else**
11 $\text{Flag} = \text{INVALID}$
12 RETURN Flag
13 RETURN

5.1.3 The Composition Procedure

The workflow is composed interactively by the composer with inputs from the user. The user specifications communicated to the composer is formalized to WCs. Using these WCs as input to the *LatticeNavigator* algorithm, the solution patterns are selected from WPL. Constraints amongst these selected patterns are stored in a relation. An entry $\text{Cons}[i, j]$ in this constraints relation represent the constraint between the patterns in indexes i and j of the selected

pattern vector

Select the solution patterns - Solution patterns are selected from WPL using *LatticeNavigator* based on user provided specifications as selection criteria

Read and validate constraints amongst selected patterns - Each Constraint $Cons[i, j]$ between selected patterns p_i and p_j that is provided by the user is validated for consistency as follows

- If the user requires $Cons[i, j]$ then find existing walks between patterns p_i and p_j
- If no walks are yielded read $Cons[i, j]$ from the user and validate it using the *ValidateConstraint* algorithm
- If existing walks are yielded then find Maximum Allowable Constraint (MAC) between p_i and p_j as follows

Initialize MAC to the power set of Allen's 13 temporal relationships. Calculate the constraint along each walk between p_i and p_j using the *ConstraintAlongWalk* algorithm. Each of these constraints is intersected incrementally with MAC. The final value from this gives the value of MAC

- Read $Cons[i, j]$ from the user. If $Cons[i, j] \subseteq MAC$ then it is validated

The pseudocode for this composition procedure is given in Algorithm 7

Algorithm 7 ComposeWG (SelPatSet, Cons)

Input: SelPatSet - Set of selected patterns, Cons - Relation storing constraints between patterns in workflow graph

Output. Cons - Relation with validated constraints of the workflow graph

```

1  MAC ← Powerset of Allen's 13 temporal relationships
2  for i= 1 to z do .
3    for j=1 to z do
4      if Cons[i,j] is required then
5        Walksi,j = FindWalks(pi, pj) {FindWalks is a function that
        returns the existing walks between patterns pi and pj }
6        if Walksi,j = φ then
7          Flag ← INVALID,
8          while ValidateConstraint(pi, pj, Cons) = INVALID do
9            read Cons[i,j],
10       else
11         for each w ∈ Walksi,j do
12           MAC = MAC ∩ ConstraintAlongWalk(w)
13         while Cons[i,j] ⊈ MAC do
14           read Cons[i,j]
```

The graph thus composed can have different execution walks based on the markings of 'start' and finish patterns. For a marking WF is added to the zeroth index of the selected pattern vector. WF is the pattern representing the workflow in its entirety.

- Marking 'start' pattern - Let p_s be the 'start' pattern. Then mark n as $c[s,0]=\{s\}$ where s and 0 are indexes of p_s and p_0 .
- Marking 'finish' patterns - Let there be n 'finish' patterns. Mark

the 'finish' pattern p_f as $c[f, 0] = \{f\}$ where f is the index of p_f in the selected pattern vector.

This marking is validated if the following holds -

- For each p_f there exists a walk from p_s to p_f
- $c[i, s] = \emptyset$ for $i \neq s$ (no in-edge to p_s , except self edges)
- $c[f, i] = \emptyset$ for $i \neq f$ (no out-edge from p_f except for self edges)

5.1.4 Illustrative Example without Security

Figure 5.1 visualizes the directed graph composed out of the requisition processing workflow in Example 1 using the *ComposeIVG* procedure. Table 5.1 gives the WPs, roles and temporal constraints involved in the workflow.

Table 5.1: WPs, roles and constraints in requisition processing workflow

WP	Roles attached	Constraints involved
Sequence1 (S1)	Requester, Approver	$S1 \rightarrow (o) \rightarrow EC1$
Exclusivechoice1 (EC1)	Approver, Procurement Officer	$EC1 \rightarrow (<) \rightarrow EC2$
Exclusivechoice2 (EC2)	Approver, Procurement Officer	$EC2 \rightarrow (<) \rightarrow S2$
Sequence2 (S2)	Store Keeper	$S2 \rightarrow (o) \rightarrow PS1$
ParallelSplit1 (PS1)	Store Keeper, Accounts Officer	$PS1 \rightarrow (o) \rightarrow SL1$
StructuredLoop1 (SL1)	Approver, Accounts Officer	$SL1 \rightarrow (d_i, f_i) \rightarrow EC1$
Exclusivechoice3 (EC3)	Approver, Accounts Officer	$EC3 \rightarrow (o) \rightarrow S3$
Sequence3 (S3)	Accounts Officer	

Above WPs namely Sequence, ExclusiveChoice, ParallelSplit and StructuredLoop are standard patterns from Van Der Aalst *et al.*'s repository.

Appendix F presents the structure of these patterns borrowed from the Van Der Aalst *et al*'s repository. As per the Definition 12 of WP, the Exclusive-Choice1 pattern is illustrated here:

T	ApproveRequisition GeneratePO NotifyRequester
R_{wp}	Approver, Procurement Officer (PO)
τ	{(Approver) (ApproveRequisition)} {(Procurement Officer) (GeneratePO NotifyRequester)}
TempConst	$ApproveRequisition \text{ -- } \neg(<) \rightarrow GeneratePO$ $ApproveRequisition \text{ -- } \neg(<)NotifyRequester$
Control-flow rules	R being the requisition processed, the control flow rules are $Approved(R) \rightarrow (GeneratePO \&\& \neg NotifyRequester)$ $Demed(R) \rightarrow (\neg GeneratePO \&\& NotifyRequester)$
REB	<div style="border: 1px solid black; padding: 5px;"> <p>(PE_1) ([1/1/2000 ∞], CurrentFinYear VH enable Approver)</p> <p>(PE_2) ([1/1/2000 ∞], MarchEnd, VH disable Approver)</p> <p>(PE_3) ([1/1/2000 ∞] CurrentFinYear, VH enable PO)</p> <p>(PE_4) ([1/1/2000 ∞] MarchEnd, VH disable PO)</p> <p>(RT_1) (enable Approver $\text{ -- } H$ enable PO)</p> </div> <p>V and H are priorities of tasks and $H \preceq VH$. CurrentFinYear = <i>all years + 4 Months \triangleright 1 Years</i> represents the set of intervals starting at the same instance as the fourth month of each year (In Gregorian calendar this is the year starting 1st of April). MarchEnd = <i>all years + 90 Days \triangleright 1 Days</i> represents the set of intervals starting at the same instance as the ninetieth day of each year. In Gregorian calendar this is the 31st March.</p>

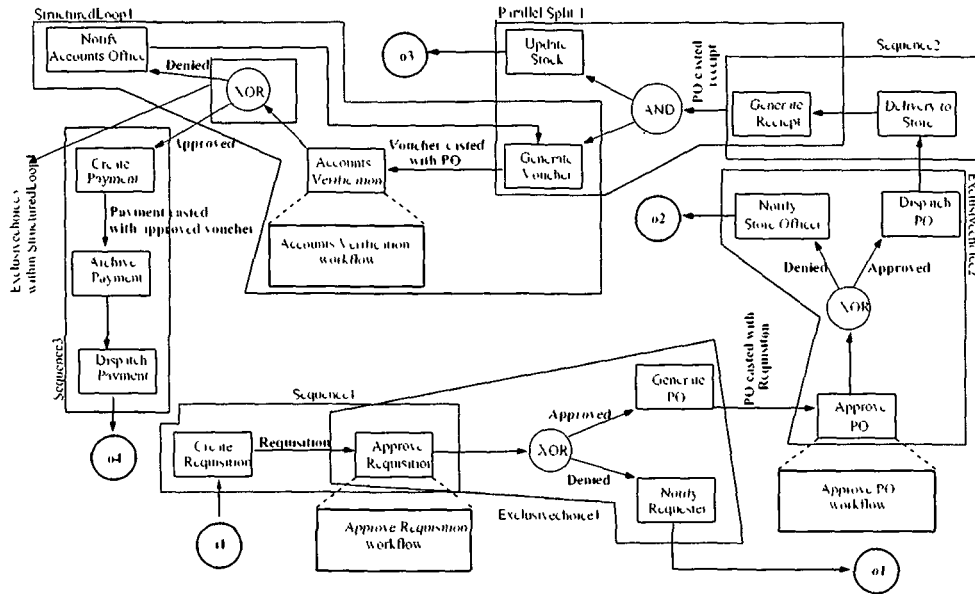


Figure 5.1 Requisition processing workflow

5.2 Composition with Security

Security is a non-functional characteristic that needs to be integrated to the workflow for increasing its robustness. In this pattern-based approach for workflow composition, a workflow is secured by integrating SPs to the constituent WPs of the workflow.

5.2.1 Secured Workflow Pattern

A SP is a task to be performed for ensuing a TE². Hence integrating SPs to a WP would result in a secured version of the WP wherein the SPs would be temporally constrained to the WP.

Definition 26 A Secured Workflow Pattern (SWP) is defined as $\langle WP, SP_{set}, C \rangle$ where

- *WP* is the workflow pattern being secured
- *SP_{set}* is the set of SPs integrated to WP
- *C* is the set of temporal constraints between patterns in *SP_{set}* and WP. SPs in *SP_{set}* are temporally independent of one another and executes in reference to WP

A SWP is essentially a WP with the tasks in the core WP and that in the integrating SPs forming the task set. The SPs in a SWP will be executed in reference to the core WP.

5.2.2 Securing a Workflow

The procedure for composition without security described in Section 5.1 could be used for securing a workflow during composition as follows

- **Constructing the SWPs from the selected WPs:** For each of the selected WP, security requirements are read from the user and

²Refer to Definition 18.

formalized as TEs. Solution SPs for these security requirements are selected from the SPL based on the formalized TEs. The SWP is thus formed by reading the temporal constraints between the selected SPs and the WP.

- **Reading and Validating constraints amongst SWPs:** The selected pattern vector in this case would hold the SWPs and the constraints relation would hold the constraints amongst the SWPs. For each entry $Cons[i, j]$ in constraints do the following
 - If $Cons[i, j]$ is required then find walks between patterns SWP_i and SWP_j
 - If no walks are yielded read and validate $Cons[i, j]$. For this following two constraints need to be validated
 - * **Constraints between SWP_a and SWP_b :** SWPs being WPs essentially the *ValidateConstraint* procedure is used here
 - * **Constraints between SPs and SWPs:** Let SP_1 be one of the SPs integrated to SWP_a . Let c_1 , c_2 and c_3 be the constraints between SP_1 and SWP_a , SWP_a and SWP_b , SP_1 and SWP_b respectively. The scenario is depicted in Figure 5.2. Here c_1 , c_2 , c_3 being temporal constraints for all possible values of c_1 and c_2 . c_3 will be given by Allen's Transitivity Table 2.2
 - If walks are yielded then find maximum allowable constraint (MAC) between SWP_i and SWP_j as follows
 - Initialize MAC to the power set of Allen's 13 temporal relation-

ships. Calculate the constraint along each walk between SWP_i and SWP_j using the *ConstraintAlongWalk* algorithm. Each of these constraints is intersected incrementally with MAC. The final value from this gives the value of MAC. Read $Cons[i, j]$. If $Cons[i, j] \subseteq MAC$ then it is validated.

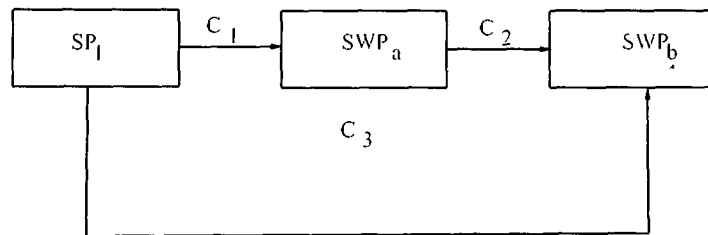


Figure 5.2 Security constraints

5.2.3 Illustrative Example with Security

Taking forward the illustration for composition without security, Table 5.3 gives the selected SPs for securing the constituent WPs in the requisition processing workflow.

The secured Sequenced WP is given in Table 5.4 as per the definition of SWP and depicted in Figure 5.3. The secured version of workflow with the SWPs is visualized in Figure 5.4.

In this chapter, an original approach for composition of a workflow as a directed graph of its patterns and temporal constraints has been given.

Table 5.3: Security requirements of Requisition Processing workflow

WP	TEs to be enforced	SPs
Sequence1	Authority	ATH,ATR
Exclusivechoice1	AP, Identity, Authority	ATH, ATR, RB
Exclusivechoice2	Authority, Integrity	ATH, ATR, RB
Sequence2		
ParallelSplit1	Authority, AP, Identity	CP
StructuredLoop1	Authority, AP, Identity	ATH,ATR
Exclusivechoice3	Authority, AP, Identity	ATH, ATR
Sequence3	Authority, AP	ATH, ATR

Table 5.4: Secured Sequence1

<i>WP</i>	- Sequence1
<i>SP_{set}</i>	- ATH for Requester (ATH_r), ATR for Requester (ATR_r), ATH for Approver (ATH_a), ATR for Approver (ATR_a)
<i>C</i>	- $ATH_r - -(<) \rightarrow Sequence1$, $ATR_r - -(<) \rightarrow Sequence1$, $ATH_a - -(d) \rightarrow Sequence1$, $ATR_a - -(d) \rightarrow Sequence1$

For constraints, it considers the set of 13 temporal relationships between a pair of ordered intervals as established in Allen's IA. A validating procedure for the constraints has also been devised in Section 5.1.2 which is also used in Chapter 6 for incorporating change into the composed workflow. The approach also gives an innovative way of composing a secured pattern-based workflow. For this, a WP is secured by integrating SPs to it such that the resultant remains essentially a WP. It is formalized as a SWP and a secured

Secured Sequence1

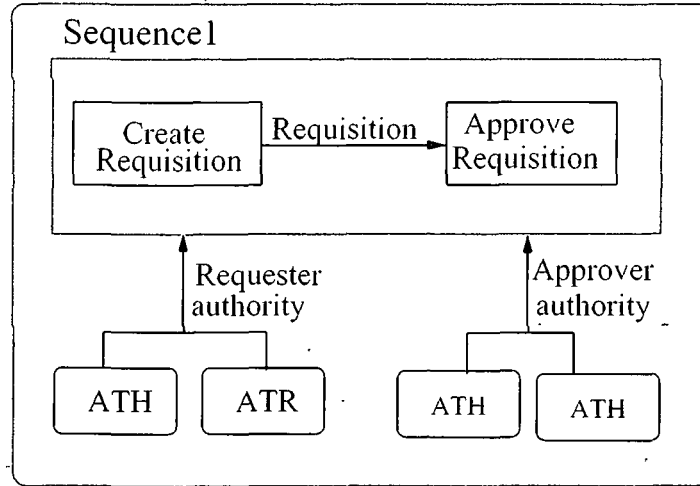


Figure 5.3: Secured Sequence1 pattern

version of the workflow is composed with these SWPs.

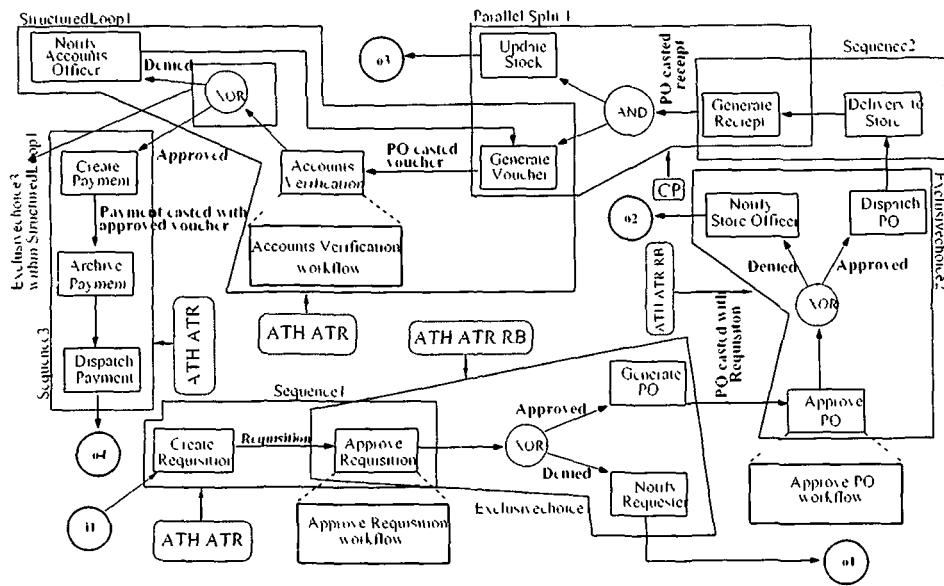


Figure 5.4 Secured Requisition processing workflow

Chapter 6

Incorporating Changes in a Workflow

Formalization of a workflow in terms of its patterns and constraints as considered in this research eases the task of maintaining the workflow. Considering a workflow at the pattern level reduces the complexity of validating changes that would be incorporated into the workflow during the maintenance phase. Change in a workflow over passage of time may occur in various forms - (a) Change in patterns (b) Change in the temporal constraints amongst patterns (c) Change in the roles attached to the patterns. This Chapter gives an approach for incorporating these changes based on Allen's IA framework and TRBAC. Here a pattern is considered to be structurally atomic in nature that could consist of other patterns as a constituent task within it. Thus a pattern could have other patterns that would refer to it during an execution. With this understanding the change incorporation approach given in this chapter considers a pattern as a *Reference Interval* and

traces a *Reference Interval Hierarchy (RIH)* from the directed graph of the workflow. Changes in patterns and temporal constraints are then incorporated into the RIH by a constraint propagation procedure. Changes in the roles attached to the patterns are incorporated by making consistency check on the REBs of the effected roles. The change incorporation process do not alter the structure of the patterns in any way.

6.1 Basic Formalization

Formalization of the change incorporation approach is done in terms of definitions and Proposition 2 that makes use of formal constructs of a workflow as done earlier in this thesis.

Definition 27 An *execution walk* $\sigma = P_1, P_2, \dots, P_n$ of a workflow is an eventually finite sequence of some or all of the constituent patterns of the workflow.

Definition 28 A *Reference Interval Hierarchy (RIH)* is a temporal hierarchy characterized by

WF a maximal interval that doesn't refer to any other interval

$R = \{r_1, r_2, \dots, r_n\}$ is the set of reference intervals constituting the hierarchy

$C' \subseteq C$ where C is the set of the temporal constraints from Allen's framework.

- For each pair of intervals (i_1, i_2) from R if $(i_1, i_2) \in RIH$ then $(i_2, i_1) \notin RIH$ and $i_1 \rightarrow i_2 = c_{i_1}$ where $c_{i_1} \in C'$

Proposition 2 Given any workflow representable as a directed graph G with temporal constraints C it is always possible to transform it to a RIH

Proof: From Proposition 1, a workflow can be represented as a directed graph of its patterns and constraints among them. Again from Definition 27 a workflow representable in terms of its pattern can in turn be considered as the sequence pattern (Pattern 1 as per Van Der Aalsts' indexing of control-flow WPs). Let $W = (WF, S, F, P, \rho, R)$ be a workflow with temporal constraints such that

WF = Pattern that represents the workflow in its entirety

S = The 'start' pattern

F = The set of 'finish' patterns

P = The set of constituent patterns including S and F

ρ = The set of temporal constraints between pairs of patterns in $P \cup WF$

Allen's representation of multiple temporal relationships between intervals is

$$T_1 - (i_1, i_2, \dots, i_n) \rightarrow T_2$$

where T_1, T_2 are intervals and i_1, i_2, \dots, i_n are possible temporal relationships between them

Using this representation and considering the fact that any set of time intervals between a start and an end time point is covered under the set of 13 temporal relationships (as has been established in Allen's transitivity table) the proposition is proved here by the process of induction

Basis: $s=WF \quad f \in F \quad s = f \quad |F|=1 \quad |P| = 1$ (There is only one pattern in W)

Here

$$(s \text{ } WF) \mid s - -(s, =, f) \rightarrow WF$$

$$(f \text{ } WF) \mid f - -(s, =, f) \rightarrow WF$$

the basis case as in figure 6 1 is transformable to an RIH as in figure 6 2

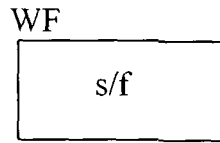


Figure 6 1 Basis of proof for Proposition 2

Case 1: $|F|=1, f \in F, s \neq f \quad |P| = 2 \quad s \notin F$ (There are two patterns in W)

Here

$$(s, WF) \mid s - -(s) \rightarrow WF$$

$$(f, WF) \mid f - -(f) \rightarrow WF$$

Case 1 as in figure 6 3 is transformable to an RIH as in figure 6 4

Case 2: $s \notin F \quad |F| \geq 1, |P| \geq 2$ (There are more than two patterns in W)

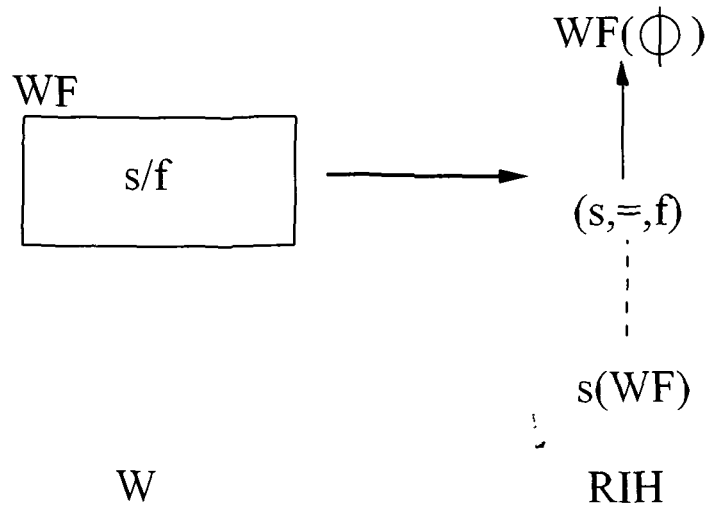


Figure 6.2: Transform of basis case of Proposition 2

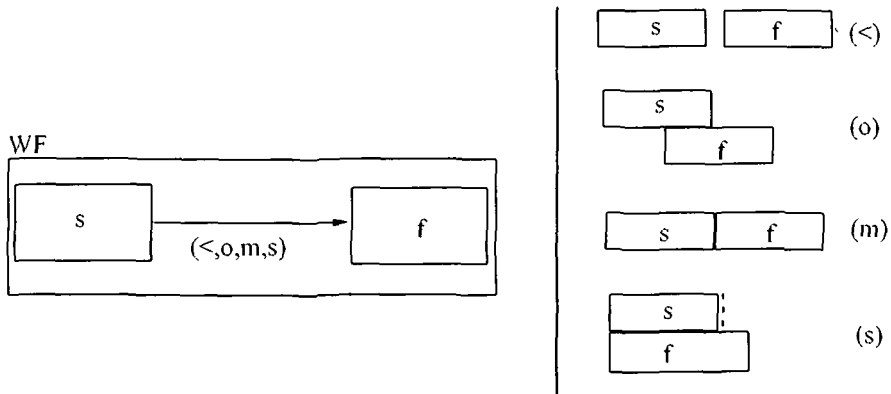


Figure 6.3: Case 1 of proof for Proposition 2

Here,

$$(s, WF) \mid s - \neg(s) \rightarrow WF$$

$$(f_j, WF) \mid f_j - \neg(f) \rightarrow WF, 1 \leq j \leq m$$

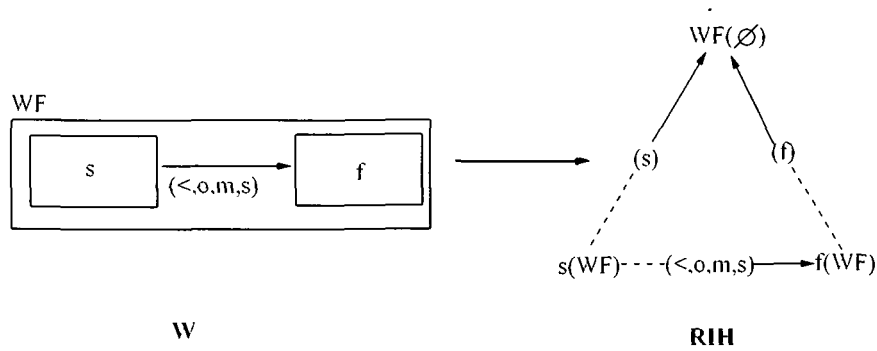


Figure 6.4 Transformation of Case 1 for Proposition 2

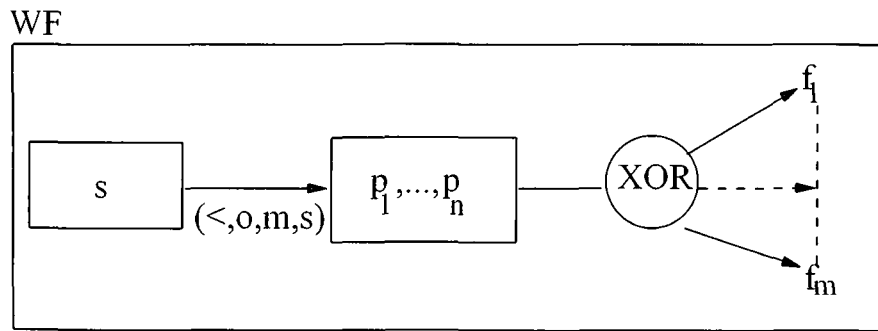


Figure 6.5 Case 2 of proof for Proposition 2

Let P_i be one of the pattern in P other than the start pattern and the finish patterns (p_i, WF) is calculated as follows

$$(s, p_i) \mid s \xrightarrow{(<, o, m, s)} p_i \text{ --- --- --- } (ii)$$

$$(s, WF) \mid s \xrightarrow{(<, o, m, s)} WF \Leftrightarrow (WF, s) \mid WF \xrightarrow{(<, o, m, s)} s \text{ --- --- --- } (iii)$$

Now, applying Allen's transitivity relations

$$\begin{aligned}
(WF, p_i) &= ((WF, s), (s, p_i)) = ((s_i), (<, o, m, s)) [from i and iii] \text{ --- } (v) \\
((s_i), (<, o, m, s)) &= ((s_i, <) \cup (s_i, o) \cup (s_i, m) \cup (s_i, s)) \text{ --- } (v) \\
(s_i, <) &= (<, o, m, d_i, f_i) \text{ --- } (vi) \\
(s_i, o) &= (o, d_i, f_i) \text{ --- } (vii) \\
(s_i, m) &= (o, d_i, f_i) \text{ --- } (viii) \\
(s_i, s) &= (s, s_i, =) \text{ --- } (ix)
\end{aligned}$$

\therefore from (iv) to (ix) above

$$\begin{aligned}
(WF, p_i) &= (<, o, m, d_i, f_i, s, s_i, =) \\
\Leftrightarrow (p_i, WF) &= (>, o_i, m_i, d, f, s_i, s, =) \text{ --- } (x)
\end{aligned}$$

From axiom 1, patterns within the workflow are execution intervals during run time, hence (x) above reduces to

$$(p_i, WF) = (d, f, s_i, s, =) \text{ --- } (xi)$$

Again, since $p_i \neq s$, $p_i \neq f_i$, $s \rightarrow WF$, $f_j \rightarrow WF$, (xi) further reduces to $(p_i, WF) \mid p_i \rightarrow (d) \rightarrow WF$

Similarly, considering the “finish” patterns one can arrive at

$$(p_i, WF) \mid p_i \rightarrow (d) \rightarrow WF$$

\therefore Case 2 as in figure 6.5 is transformable to an RIH as in figure 6.6

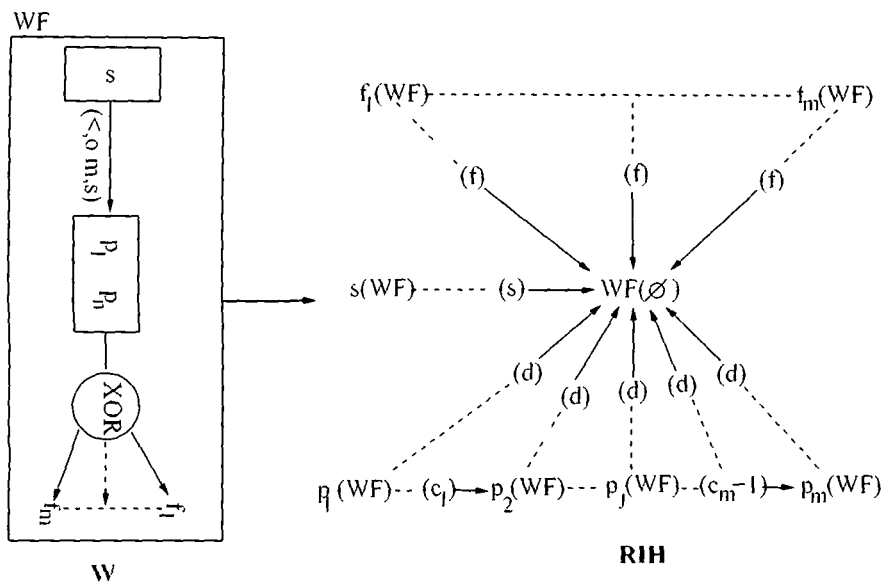


Figure 6.6 Transformation of Case 2 for Proposition 2

Thus, from basis, Case 1 and Case 2 the proposition is established through induction

6.2 Functionalities Involved

Changes that may arise in patterns in a workflow would also affect temporal constraints. Any changes in the constraints would have a cascading effect on the entire workflow since the patterns are transitively related to one another by the constraints. Also change in patterns would bring about changes in roles. These changes are accommodated into the workflow graph as follows:

1. The graph of the workflow being maintained is transformed into a RIH by the *Transform* procedure.

- 2 Changes in the form of new patterns being added and existing constraints being changed are incorporated by the *IntroduceNewPattern* and *ChangeConstraint* procedures. These procedures take care of the cascading effect of the changes being incorporated in a path-consistent manner. Changes in roles resulting from the incorporated changes are introduced by the process given in Section 6.3
- 3 Once the changes are successfully incorporated to the RIH, the updated RIH is transformed back to its directed graph form by the *Inverse-Transform* procedure. The updated workflow goes into execution in an evolved form and further changes are incorporated similarly as in the above steps.

6.2.1 Building the *Reference Interval Hierarchy*

Since the RIH is a hierarchy of intervals, each pattern interval in it will be related to its *Reference Intervals* by one or more of the following temporal relations - (*during(d)*, *start(s)*, *finish(f)*, *equal(=)*). To keep track of the patterns that have been selected in RIH from \mathcal{W} , each pair of (pattern interval) is marked with a pair of flags (skel, stub) where skel flags the selected pattern in \mathcal{W} and stub flags the corresponding interval in the RIH. The RIH could be built from the workflow graph by the following procedure:

Step 1 - Mark all the patterns that are related to WF by one or more of (s d f =). Let these patterns be given by $P_{initial}$

Step 2 - For each pattern in $P_{initial}$, recursively back trace the workflow

graph. In each step of the back tracing, Let P_{ref} be the *Reference Interval* pattern of P_{cur} - the pattern being visited currently and P_{adj} be the set of adjacent patterns directly connected to P_{cur} . To start with ϕ is the *Reference Interval* pattern of WF.

Step 2.1 - Make P_{cur} the *Reference Interval* of those elements in P_{adj} which are related to P_{cur} by one or more of (s d f =)

Step 2.2 - Label the edges amongst those elements in P_{adj} selected in Step 2.1 by the Constraints amongst them.

Step 2.3 - Continue with the back tracing with those elements in P_{adj} that have outgoing edges. If no such elements exist or all such elements have been exhausted then continue with the back tracing with the remaining adjacent elements of P_{ref} yet to be explored. This process continues until all the patterns in the sub graph rooted at the current pattern in $P_{initial}$ has been explored completely.

Step 3 - The RIH will be built once all the patterns in $P_{initial}$ has been explored as in Step 2.

The pseudocode for this *Transform* function is given in Algorithm 9. *Transform* makes a call to the recursive procedure *ExploreSubGraph* that back traces the graph recursively.

Algorithm 8 Exploresubgraph(P_{cur} , W_{pat} , $Cons$, RIH_{pat} , $TCons$)

Input P_{cur} - Current pattern in graph being visited W_{pat} - Patterns in graph
 $Cons$ - Relation holding constraints amongst patterns in graph RIH_{pat}
-Selected patterns in RIH $TCons$ - Relation holding constraints amongst
selected patterns in RIH

Output: RIH_{pat} - Array holding reference intervals of selected patterns in RIH

```

1  $P_{adj} \leftarrow$  Adjacent patterns of  $P_{cur}$ 
2 for each  $P_j \in P_{adj}$  do
3   Visited( $P_j$ )
4   if Edge( $P_{cur}, P_j$ ) is one or more of (s,d f,=) then
5      $l =$  index of  $P_{cur}$ ,  $m =$  index of  $P_j$ 
6     Add  $Cons[l, m]$  to  $TCons$  { $Cons[l, m]$  is the constraint between  $P_{cur}$  and
        $P_j$ }
7      $P_{j,ef} \leftarrow P_{cur}$  { $P_{j,ef}$  is reference interval of  $P_{cur}$ }
8     Add( $P_{j,ef}$   $RIH_{pat}$ )
9     if  $P_{cur}$  is not selected then
10      Add( $P_{cur}$   $RIH_{pat}$ ),
11  $P_{out} \leftarrow$  Elements of  $P_{adj}$  having outgoing edge
12 if  $P_{out} = \phi$  or each element in  $P_{out}$  marked visited then
13   RETURN
14 else
15    $P_{next} \leftarrow$  Next element in  $P_{out}$  not visited yet
16   ExploresubGraph( $P_{next}$   $W_{pat}$ ,  $Cons$   $RIH_{pat}$   $TCons$ )

```

Algorithm 9 Transform (W_{pat} , Cons)

Input: W_{pat} - Array holding patterns in workflow graph Cons - Relation holding constraints amongst patterns in W_{pat}

Output: RIH_{pat} - Array holding reference intervals of selected patterns in RIH
TCons - Relation holding constraints amongst patterns in RIH_{pat}

- 1 NotSel(W_{pat}) {The patterns in graph are initially marked as not selected}
- 2 $P_{initial} \leftarrow P_1, P_2, \dots, P_n$ {Patterns related to WF by one or more of (s, d f =)}
- 3 $P_{ref} = \phi$ {Reference Interval of the whole workflow WF}
- 4 Append (P_{ref} , RIH_{pat})
- 5 for $i=1$ to n do
- 6 Append constraint between P_{ref} and P_i to TCons
- 7 Visited(P_i), Selected(P_i) { P_i is marked visited and selected in W_{pat} with *skel* and *stub* flags}
- 8 ExploreSubGraph(P_i , W_{pat} , Cons, RIH_{pat} , TCons)

6.2.2 Incorporating Change

In the scope of this research, changes in the form of a new pattern being introduced and change in the value of an existing constraint are considered. For both the cases, the issue to be taken care of is whether the rest of the constraints in the RIH would remain consistent or not after introduction of the changes. For this, the approach taken here first validates the cascading effect of the change along all the effected paths in the RIH and then introduces the desired change. This makes the changes path-consistent in nature.

- **Introducing a new pattern** - This would effectively mean that a new walk is to be established between two existing patterns which would

have only this new pattern as an intermediate vertex. If A and B are the two existing patterns and C is the new pattern then C could be consistently introduced to the workflow as follows.

Let i and j be the lexicographic indexes of patterns A and B. A consistent walk P between A and B would be of the form $P^{ij}, P^{ij_{i+1}}, P^{ij_{i+1-1}}, P^{ij_{i+k}}, P^{ij_{i+k+1}}, P^{ij_{i+j-1}}, P^{ij_{i+j}}$ where any two consecutive P^{ij} s are explicitly connected by a constraint. The transitivity constraint T along the walk P is calculated by the *ConstraintAlongWalk* procedure. Similarly the constraints along all other walks between A and B are calculated and their intersection is considered as R. After calculating R in this manner, constraint C1 between A and C and C2 between C and B are read. C1 would be validated iff $C1 \subseteq R$ and C2 will be validated iff $CalTransCons(C1, C2) \subseteq R$. Algorithm 10 lists the pseudocode for this *IntroduceNewPattern* procedure.

- **Change in an existing constraint** - Let C be the constraint between patterns P_{l-1} and P_k and C is to be changed. This will affect all the walks involving P_{k-1} and P_k . Let $P = P_1, P_2, \dots, P_{k-1}, P_k, P_{k+1}, \dots, P_n$ be one such walk. Let $R_{o, \eta}$ be the constraint along walk P. R_{k-1} be the constraint along walk up to P_{l-1} . Calculate R_k the constraint along the walk starting from P_{k-1} with the changed value of C. Let R_{new} be the new constraint along walk P with the changed value of C. R_{new} is calculated as $CalTransCons(R_{k-1}, R_k)$. Here the changed value of C will be consistent and validated iff $R_{new} \subseteq R_{o, \eta}$. The pseudocode for this *ChangeConstraint* procedure is given in Algorithm 11.

Algorithm 10 IntroduceNewPattern ($Pat_a, Pat_b, Pat_{new}, C_1, C_2, RIH_{pat}, TCons$)

Input: Pat_a, Pat_b, Pat_{new} - Pat_{new} is the new pattern to be inserted between Pat_a and Pat_b C_1 - Constraint between Pat_a and Pat_b C_2 - Constraint between Pat_{new} and Pat_b RIH_{pat} - A DAG holding the patterns in RIH $TCons$ - Relation holding the constraints amongst the patterns in the RIH of the workflow graph

- 1 $i \leftarrow$ Index of Pat_a in RIH_{pat}
- 2 $j \leftarrow$ Index of Pat_b in RIH_{pat}
- 3 Conwalks = Consistent walks between A and B {Found by any straightforward graph search algorithm like BFS Each element $CP \in$ Conwalks is of the form $P^{ij}_i, P^{ij}_{i+1}, \dots, P^{ij}_{i+k-1}, P^{ij}_{i+k}, P^{ij}_{i+k+1}, \dots, P^{ij}_{i+j-1}, P^{ij}_{i+j}$ }
- 4 $R \leftarrow \varepsilon$ { ε is the complete set of 13 temporal relations from Allen's IA R represents constraints along all consistent walks between A and B}
- 5 **for** each $CP \in$ Conwalks **do**
- 6 $P_{first} \leftarrow$ FirstPat(CP)
- 7 $P_{last} \leftarrow$ LastPat(CP) {FirstPat and LastPat are functions that return the first and last patterns of a walk respectively}
- 8 $T \leftarrow$ ConstraintAlongWalk($P_{first}, P_{last}, TCons$)
- 9 $R \leftarrow R \cup T$
- 10 **if** $C_1 \subseteq R$ and $CalTransCons(C_1, C_2) \subseteq R$ **then**
- 11 Introduce Pat_{new} between Pat_a and Pat_b by adding Pat_{new} to RIH_{pat} {This creates a walk Pat_a, Pat_{new} and Pat_b in the workflow graph}

Algorithm 11 **ChangeConstraint** (C_{new} , Pat_a , Pat_b , RIH_{pat} , **TCons**)

Input: Pat_a Pat_b - patterns between which constraint is to be changed C_{new} -
 New constraint between Pat_a and Pat_b RIH_{pat} - Array holding the patterns
 in RIH **TCons** - Relation holding constraints amongst the patterns in the
 RIH of the workflow graph

- 1 $C_{orig} \leftarrow$ Original constraint between patterns Pat_a and Pat_b
- 2 **Effecteds** \leftarrow walks involving both P_{k-1} and P_k in workflow graph
- 3 **ValidFlag** \leftarrow **TRUE** {Flag for indicating validity of new constraint
 being introduced}
- 4 **for each** $P \in$ **Effecteds** **do**
- 5 $R_{orig} =$ Original constraint along P $R_a =$ constraint along P up to P_a R_b
 = constraint along P starting from P_a with new constraint C_{new} between P_a
 and P_b {Here *ConstraintAlongWalk* procedure is used for calculating R_{orig} ,
 R_a and R_b }
- 6 $R_{new} \leftarrow$ **CalcTransCons**(R_a, R_b)
- 7 **if** $\neg(R_{new} \subseteq R_{orig})$ **then**
- 8 **ValidFlag** \leftarrow **FALSE**
- 9 **RETURN FALSE,**
- 10 **if** **ValidFlag** == **TRUE** **then**
- 11 $C_{orig} \leftarrow C_{new}$ {Replace original constraint by new constraint}
- 12 **RETURN TRUE**

6.2.3 Inverse Transforming *Reference Interval Hierarchy* to Workflow Graph

The inverse transform function takes the RIH with changes incorporated and
 outputs the equivalent workflow graph. It is parameterized with a starting

Reference Interval, $Start_{RI}$ A call is made to the inverse transform function with $WF(\phi)$, the pattern representing the whole workflow that doesn't refer to any other interval. The procedure works as follows

- Start with $Start_{RI}$
- Let Int_{cur} be the current interval in the hierarchy being explored and Int_{ref} be the intervals referencing Int_{cur}
- Let I_{e11st} be the latest interval explored which had an equivalent pattern in W . Initialize it to NULL
- For each interval $I \in Int_{ref}$ do the following
 - If this is a new interval and I_{e11st} is not null, add the pattern for this Interval in W . This pattern will be connected to the corresponding pattern of I_{e11st} by the edge between the two in RIH. Check for a new interval can be made by using the (skel stub) flags
 - If this is not a new interval then check whether the edge between I_{e11st} and I in RIH is similar to the corresponding edge in W . If not then update the edge in W accordingly
 - Make I_{e11st} the next element in I_{ref} . If I_{e11st} is found null then the procedure completes and W is updated with the incorporated changes. If I_{e11st} is not null then repeat the inverse transform recursively with I_{e11st} as $Start_{RI}$

Algorithm 12 gives the pseudocode for this inverse *InverseTransform* procedure

Algorithm 12 InverseTransform (WF, RIH_{pat} , TCons, W_{pat} , Cons)

Input: RIH_{pat} - Array holding patterns in RIH TCons - Relation holding constraints in RIH W_{pat} - Array holding patterns in W, Cons - Relation holding constraints in W

Output: Updated W_{pat} and Cons,

```

1  $Int_{cur} \leftarrow WF$   $Int_{ref} \leftarrow$  Intervals referencing  $Int_{cur}$   $I_{first} \leftarrow NULL$   $\{I_{first}$ 
   represents the latest interval explored and that has a corresponding pattern
   in W $\}$ 
2  $I \leftarrow First(Int_{ref})$   $\{First$  returns the first element in  $Int_{ref}$   $\}$ 
3 while  $I \neq NULL$  do
4   if  $InWG(I) = FALSE$  then
5      $I_{new} \leftarrow I$   $\{InWG$  is a function that searches for the equivalent skel of  $I$ 
      in W and returns true if found $\}$ 
6     if  $I_{first} \neq NULL$  then
7        $AddEdge_W(I_{first}, I_{new}, Edge(I_{first}, I_{new}))$   $\{AddEdge_W$  adds the new
        pattern into the graph $\}$ 
8     else
9        $I_{prev} \leftarrow I_{first}$ ,  $I_{cur} \leftarrow I$ 
10    if  $Edge_W(I_{prev}, I_{cur}) \neq Edge_{RIH}(I_{prev}, I_{cur})$  then
11       $Edge_W(I_{prev}, I_{cur}) \leftarrow Edge_{RIH}(I_{prev}, I_{cur})$   $\{Edge_W$  and  $Edge_{RIH}$  are
        functions returning constraints from W and RIH $\}$ 
12     $I_{first} \leftarrow Next(Int_{ref})$ ,  $I \leftarrow I_{first}$ 
13    if  $I_{first} = NULL$  then
14      RETURN
15    else
16       $InverseTransform(I_{first}, RIH_{pat}, TCons, W_{pat}, Cons)$ 

```

6.3 Change in Roles Arising from Changes in Workflow

Change in roles that may arise due to accommodation of changes in workflow can be checked for consistency by checking the REBs that gets introduced or effected by the change. Change in a workflow can either be change in an existing constraint or introduction of a new pattern. Change in an existing constraint will have no effect on the roles since these changes will not affect the Periodic Events (PEs) in the REBs attached to the WPs whose constraints have been changed. Whereas introduction of a new pattern can bring in two possible changes - New roles introduced in the REB of the new pattern and existing roles assigned to the new pattern. The whole scenario is depicted in Figure 6.7

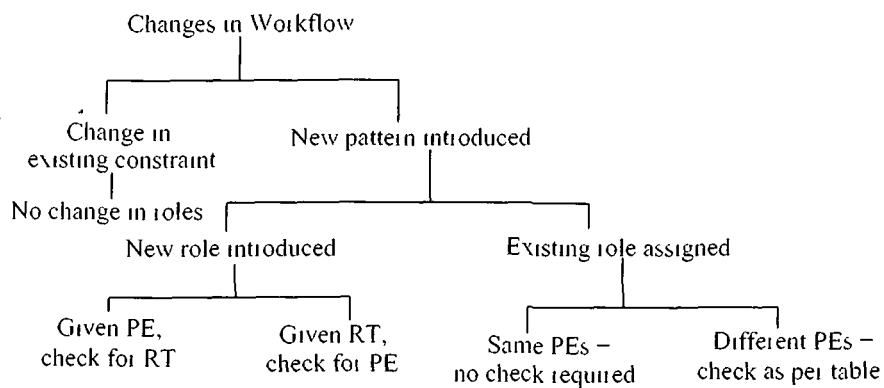


Figure 6.7 Role change scenario

6.3.1 Introduction of a New Role

REB of a pattern have PEs and RTs which involve enabled and disabled status of the roles attached to the pattern. For a new role being introduced due to introduction of a new pattern, the REB of the new pattern would be having PE or RT or both involving the new role. In this case, consistency check could be made as follows:

- Given the PEs, consistency check is made for the status of the new role in the RTs.
- Given the RTs, consistency check is made for the status of the new role in the PEs.

An illustration of this is given in Section 6.5.3.

6.3.2 Reassignment of an Existing Role

In this case, the periodic events in the REB of the new pattern involving the existing role is compared with the periodic events in the REBs of the existing patterns involving the same role. No consistency check would be required if the periodic events are same. Consistency check on the status of the roles could be done as per Table 6.1 if the periodic events are different.

Here

- ER : Existing role
- $PE_{exist}(I_{exist}, PE_{exist})$: Periodic event in existing pattern involving ER where I_{exist} is the interval and PE_{exist} is the periodic expression.

Table 6.1 Consistency check table for role triggers in REB

Intervals	Periodic expressions	status of ER	feasibility
same	same	same	allowed
same	same	different	not allowed
same	different	same	allowed
same	different	different	allowed only if calendars in the two expressions are disjoint
different	same	same	allowed
different	same	different	allowed only if $I_{exist} - (\langle m \rangle) - I_{ren}$
different	different	same	allowed
different and disjoint	different	different	Allowed
different and not disjoint	different and disjoint	different	allowed
	different and not disjoint	different	Allowed only if different calendars in $PE_{Exist} \cap PE_{New}$

- $PE_{new}(I_{new} PE_{New})$ Periodic event in existing pattern involving ER where I_{new} is the interval and PE_{New} is the periodic expression

6.4 Change in Security Arising from Changes in Workflow

Security aspects has been integrated to a workflow during composition by transforming WPs to SWPs. A SWP is essentially a WP as has been formalized in Definition 26. It has a core WP and SPs related to it by temporal

constraints. Therefore, security related changes could be of two forms - introduction of a SP to a SWP, change in the constraint between a SP and its SWP.

6.4.1 Introduction of a New Security Pattern

A new SP can be introduced to the workflow by integrating it to a particular SWP. SPs in a SWP execute independently of one another. Thus, this will have just one implication to be taken care of - how will this new SP be constrained with other SWPs directly connected to its SWP. Further cascading effect of this change could be avoided if the new SP is introduced in such a manner that the constraints between its SWP and those directly connected to it remains unchanged. This can be achieved by the process taken for validating a constraint between a SP and SWP during workflow composition.

6.4.2 Change in Existing Constraint

In this case, the required change will have a transitive effect on the constraints amongst the effected SWP and the SWPs directly connected to it. Thus, without changing the constraints amongst the SWPs, the required change in constraint amongst the SP and its SWP could be validated using Allen's Transitivity table and incorporated accordingly. This approach is again similar to the one taken during composition with security.

6.5 Illustrating the Approach

Figure 6.8 gives the equivalent RIH transformed from the directed graph in Figure 5.1 of the requisition processing workflow given in Example 1. Given a pattern interval Pat_{int} and its reference interval Ref_{int} in the RIH it is represented as $Pat_{int}(Ref_{int})$. All such Pat_{int} are connected to their respective Ref_{int} by one or more of the following temporal relations - (d, s, f) . In this RIH, $ReqProc(\phi)$ represents the whole requisition processing workflow as an interval that doesn't refer to any other intervals. Thus this forms the root of the RIH. $Exclusivechoice3(StructuredLoop1)$ $ExclusivechoiceN(StructuredLoop1)$ are the multiple instances of $Exclusivechoice3$ from within $StructuredLoop1$. Therefore each of these instances are related to its reference interval $StructuredLoop1(ReqProc)$ by the temporal constraint (d) .

6.5.1 Illustrating Workflow Pattern

The $Exclusivechoice2$ pattern in the requisition processing workflow is formalized according to Definition 12 in Table 6.2. The equivalent RIH of this pattern is depicted in Figure 6.9.

[127] is the only other attempt at formalizing the temporal characteristics of WP using Allen's IA. It represents the first 20 control-flow WPs from Van Der Aalst *et al*'s repository. This representation successfully covers the control-flow of the constituent tasks in the patterns by a set of formulas in terms of the end points of the tasks. However, the aspect of roles attached to these WPs while constituting a workflow, temporal constraints amongst the

WPs in the workflow and that a WP could form a hierarchy of other patterns hasn't been taken care of. These issues have been successfully covered in the formal approach given in this research work.

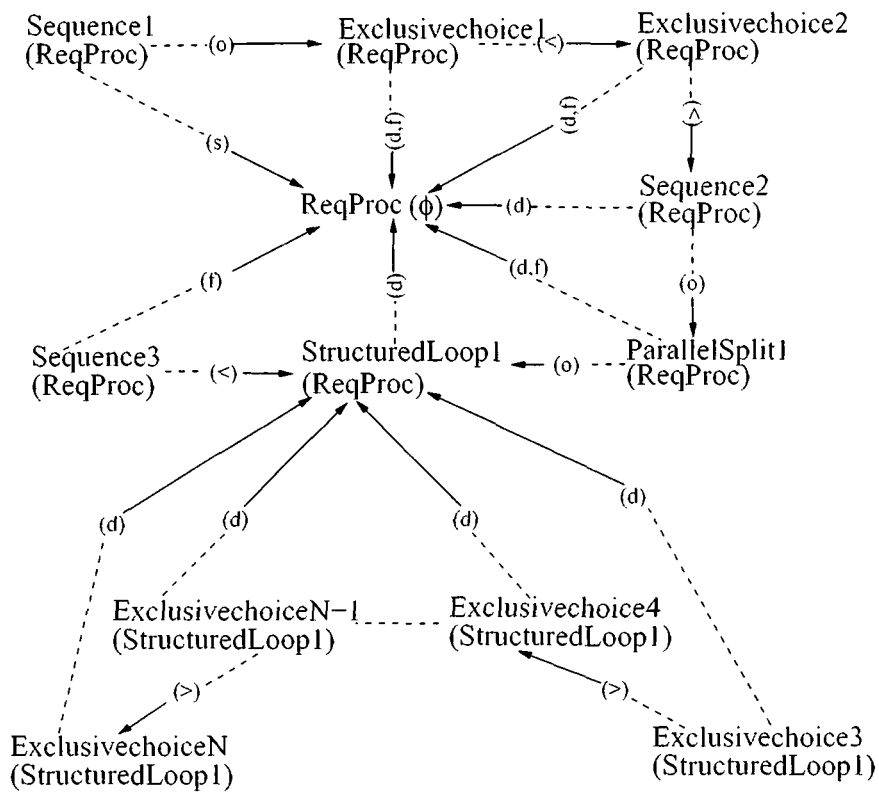


Figure 6.8: Interval hierarchy of the requisition processing workflow

Table 6.2 Exclusivechoice2

T	ApprovePO DispatchPO Notify Store Keeper
R_{i,p}	Approver Procurement Officer (PO)
τ	{(Approver) (ApprovePO)} {(ProcurementOfficer) (DispatchPO NotifyStorekeeper)}
TempConst	ApprovePO--(<) — DispatchPO ApprovePO--(<)NotifyStorekeeper
Control-flow rules	PO being the Purchase Order generated on approved requisition the control flow rules are Approved(PO) — (DispatchPO) Denied(PO) (¬DispatchPO && NotifyRequester)
R_{EB}	<div style="border: 1px solid black; padding: 5px;"> <p>(PE₁) ([1/1/2000 ∞] CurrentFinYear VH enable Approver) (PE₂) ([1/1/2000 ∞] MarchEnd VH disable Approver) (PE₃) ([1/1/2000 ∞] CurrentFinYear VH enable PO) (PE₄) ([1/1/2000 ∞] MarchEnd VH disable PO) (RT₁) (enable Approver — H enable PO)</p> </div>
V_{i,d}	4

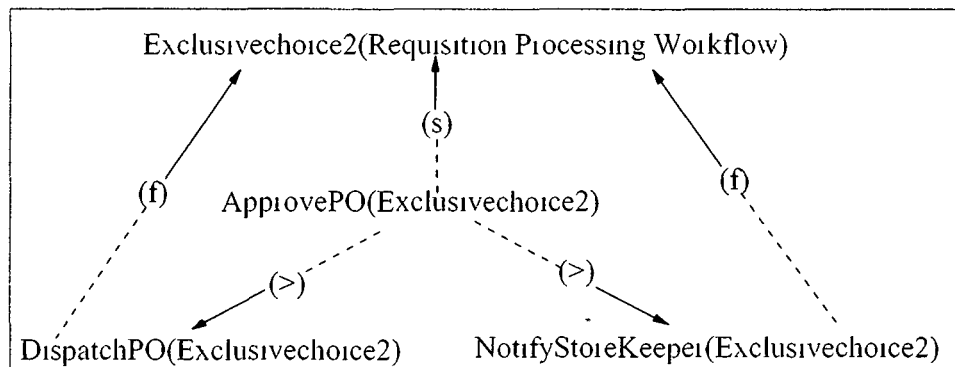


Figure 6.9 Interval hierarchy of Exclusivechoice2

6.5.2 Illustrating Changes Introduced in Workflow

Introduction of a New Pattern

In Figure 6.10, new pattern Sequence4 shown in Table 6.3 is added between the patterns Exclusivechoice1 and Exclusivechoice2 in the requisition pro-

cessing workflow. Hence, $C = (<)$ is the existing constraint between Exclusivechoice1 and Exclusivechoice2. The condition for consistency and validity of Sequence4 being introduced is $CalTransCons(C1, C2) \subseteq C$, where $C1 =$ constraint between Exclusivechoice1 and Sequence4, $C2 =$ constraint between Sequence4 and Exclusivechoice2. Making $C1 = (<)$ and $C2 = (<)$ as shown in Figure 6.10, introduction of Sequence4 can be made consistent and validated.

Table 6.3 Sequence4

T	EnableApprovalWorkflow, AuthorizePOApprover
R_{wp}	Workflow-Enable!, Authorizer
τ	{(Workflow-Enable!) (EnableApprovalWorkflow)} {(Authorizer) (AuthorizePOApprover)}
TempConst	EnableApprovalWorkflow -- (in <) -- AuthorizePOApprover
Control-flow rules	POApprover being the approver of the PO, the control-flow rules are Unauthorized(POApprover) — (Disabled(Exclusivechoice2))
$\mathcal{R}\mathcal{E}\mathcal{B}$	<div style="border: 1px solid black; padding: 5px;"> <p>(PE₁) ([1/1/2000, ∞] CurrentFinYear, \H enable Workflow-Enable!)</p> <p>(PE₂) ([1/1/2000, ∞] MarchEnd, \H disable Workflow-Enable!)</p> <p>(PE₃) ([1/1/2000, ∞] CurrentFinYear, \H enable Authorizer)</p> <p>(PE₄) ([1/1/2000, ∞] MarchEnd, \H disable Authorizer)</p> <p>(RT₁) (enable Workflow-Enable! — H enable Approver)</p> <p>(RT₂) (disable Workflow-Enable! — H disable Approver)</p> <p>(RT₃) (disable Workflow-Enable! — H disable Authorizer)</p> </div>
V_{index}	1

Change in Value of an Existing Constraint

Let the constraint between the patterns Exclusivechoice1 and Exclusivechoice2 be changed from $(<)$ to (m) such that

- R_{new} = new constraint along a walk having both these patterns

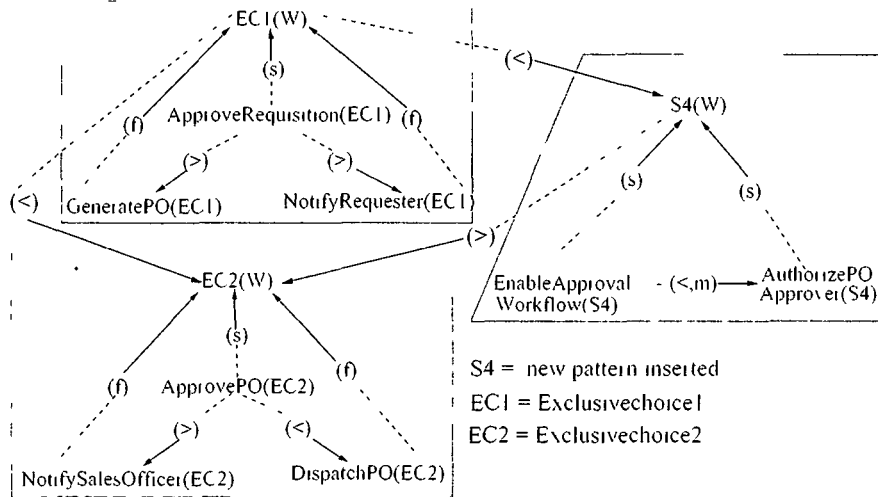


Figure 6 10 Introducing a new pattern

- R_{orig} = original constraint along this walk before the change was introduced

The change will be validated if the condition $R_{new} \subseteq R_{orig}$ holds for each and every walk having the start pattern Sequence1 as the initial node and having both the patterns in it. From Figure 6 8, there are four walks involving Exclusivechoice1 and Exclusivechoice2

walk 1 Sequence1 - (s o f_i) → Exclusivechoice1 - (<) → Exclusivechoice2 - (d f) → ReqProc

walk 2 Sequence1 - (s o f_i) → Exclusivechoice1 - (<) → Exclusivechoice2 - (<) → Sequence2 - (d) → ReqProc

walk 3 Sequence1 - (s o f_i) → Exclusivechoice1 - (<) → Exclusivechoice2 - (<) → Sequence2 - (s o f_i) → ParallelSplit - (d f) → ReqProc

walk 4 Sequence1 - (s o f_i) → Exclusivechoice1 - (<) → Exclusivechoice2 - (<) → Sequence2 - (s o f_i) → ParallelSplit - (m s) → StructuredLoop1 - (d) →

R_{o,q}P₁ or

Let Exclusivechoice1 = P_{k-1} , Exclusivechoice2 = P_k . Here $R_{k-1} = (s, o, f_i)$ is the constraint up to P_{k-1} which remains same for all the walks. R_k is the constraint from P_{k-1} in each walk with the changed value in the constraint between P_{k-1} and P_k . Consistency check is done for the walks as follows

walk 1:

$$\begin{aligned}
 R_{o,q} &= \text{CalTransCons}(\text{CalTransCons}((s \ o \ f_i) (<)) (d \ f)) \\
 &= \text{CalTransCons}((<) (d \ f)) = (< \ o \ m \ d \ s) \\
 R_l &= \text{CalTransCons}((m) (d \ f)) = (o \ d \ s \ <) \\
 R_{new} &= \text{CalTransCons}((R_{k-1}) (R_k)) \\
 &= \text{CalTransCons}((s \ o \ f_i) (o \ d \ s \ <)) = (< \ o \ m \ d \ s)
 \end{aligned}$$

walk 2:

$$\begin{aligned}
 R_{o,q} &= \text{CalTransCons}(\text{CalTransCons}(\text{CalTransCons}((s \ o \ f_i) (<)) (<)) (d)) \\
 &= \text{CalTransCons}(\text{CalTransCons}((<) (<)) (d)) = \text{CalTransCons}((<) (d)) = (< \\
 &\quad o \ m \ d \ s) \\
 R_k &= \text{CalTransCons}(\text{CalTransCons}((m) (<)) (d)) = \text{CalTransCons}((<) (d)) \\
 &= (< \ o \ m \ d \ s) \\
 R_{new} &= \text{CalTransCons}((R_{l-1}) (R_l)) = \text{CalTransCons}((s \ o \ f_i) (< \ o \ m \ d \ s)) = \\
 &(< \ o \ m \ d \ s)
 \end{aligned}$$

With similar computations for walks 3 and 4 it is seen that $R_{new} \subseteq R_{o,q}$ for all the walks and hence the change is consistent

6.5.3 Illustrating Changes on Roles

As discussed in section 6.3, changes on roles could happen only in the case of new patterns being introduced. These changes could be either new roles introduced in the REB of the new pattern or existing roles assigned to the new pattern.

New Roles Introduced with a New Pattern

New roles introduced along with new pattern sequence4 are Workflow-Enabler and Approver. From Table 6.3 REB of Sequence4 is

(PE_1) $\{(1/1/2000 \infty) \text{CurrentFinYear} \setminus H \text{ enable Workflow-Enabler}\}$
(PE_2) $\{(1/1/2000 \infty) \text{MarchEnd} \setminus H \text{ disable Workflow-Enabler}\}$
(FE_3) $\{(1/1/2000 \infty) \text{CurrentFinYear} \setminus H \text{ enable Approver}\}$
(PE_1) $\{(1/1/2000 \infty) \text{MarchEnd} \setminus H \text{ disable Approver}\}$
(RT_1) $(\text{enable Workflow-Enabler} \rightarrow H \text{ enable Approver})$
(RT_2) $(\text{disable Workflow-Enabler} \rightarrow H \text{ disable Approver})$
(RT_3) $(\text{disable Workflow-Enabler} \rightarrow H \text{ disable Approver})$

This REB defines role triggers that involves roles from REB of another pattern Exclusivechoice1. From Table 5.2 the REB of Exclusivechoice1 is

(PE_1) $\{(1/1/2000 \infty) \text{CurrentFinYear} \setminus H \text{ enable Approver}\}$
(FE) $\{(1/1/2000 \infty) \text{MarchEnd} \setminus H \text{ disable Approver}\}$
(PE_3) $\{(1/1/2000 \infty) \text{CurrentFinYear} \setminus H \text{ enable PO}\}$
(PE_1) $\{(1/1/2000 \infty) \text{MarchEnd} \setminus H \text{ disable PO}\}$
(RT_1) $(\text{enable Approver} \rightarrow H \text{ enable PO})$

Periodic events of Sequence4 involves only the newly introduced roles. Hence consistency needs only to be checked for the role triggers in the REB of Sequence4.

RT_1 - Status of the two roles Workflow-Enabler and Approver are same as their status in the similar periodic expressions they are involved. PE_1

in REB of sequence4 and PE_1 in REB of Exclusivechoice1. Hence RT_1 is consistent.

RT_2 - Similarly, consistency holds for RT_2 where the similar periodic expressions are PE_2 in REB of Sequence4 and PE_2 in REB of Exclusivechoice1.

RT_3 - It is a trivial case as it involves different roles for the same periodic events and from the same REB.

Existing Roles Assigned to a New Pattern

Let the Approver role of Exclusivechoice1 pattern be reassigned to new pattern Sequence4 and the REB with this existing role be

(PE_1) $([1/1/2000 \infty]$ CurrentEnableYear VH enable Approver)
(PE_2) $([1/1/2000 \sim]$ MarchEnd VH disable Approver)

The consistency check in this case can be done from Table 6.1. The REBs of Exclusivechoice1 and Sequence4 involve the same intervals and same status of the existing role i.e. Approver while only the periodic expression is different. Hence, from the third case in Table 6.1, the REB is consistent.

In this chapter, an innovative approach has been taken for incorporating changes in a workflow composed as a directed graph of its patterns and temporal constraints amongst them. For the first time, the concept of *Reference Interval Hierarchy* based on the Reference Interval construct of Allen's IA has been formalized. The approach works out three functionalities for incorporating changes - A *Transform* function that traces a RIH from

a workflow W , *IntroduceNewPattern* and *ChangeConstraint* procedures that incorporates changes into the RIH in a path-consistent manner. An *Inverse-Transform* procedure that transforms back the updated RIH to its directed graph form. These functionalities take care of the changes in the patterns and constraints. Changes in the roles affected by these changes are checked for consistency and validity by checking the REB of the affected patterns. For consistency check of the changes in the roles, Table 6.1 has been worked out which tabulates the feasibility of all possible combination of intervals, periodic expressions and status of a role in the RTs of an REB having the affected role. The originality of this approach stems from the formalization of RIH transformation of a workflow as a RIH, use of constraint propagation for incorporating change and its cascading effect in a workflow, inverse transforming a RIH to its directed graph form and the consistency check table for role triggers of REBs attached to patterns.

The change incorporation approach as taken in this chapter serves the purpose of maintenance of a workflow over passage of time. This and the other two approaches in Chapter 4 and Chapter 5 fulfill the overall approach of composing and maintaining a workflow over passage of time so that the workflow is allowed to evolve in a robust way with changes being incorporated in a valid and consistent manner.

Chapter 7

Summary and Future

Directions

7.1 Summary of the Work Done

This research work is on a pattern-based approach for composing and maintaining a workflow in a formal manner with a modular architecture. The architecture consists of three modules - formal organization of patterns as lattices, composition of workflow as a directed graph of patterns and temporal constraints amongst them, incorporation of changes into the workflow over passage of time. Pattern lattices of WPs and SPs are built using procedures *LatticeGenerator* and *LatticeNavigator*. Composition of the workflow is done making use of *ComposeWG* procedure and validating procedures *ConstraintAlongWalk* and *ValidateConstraint*. For incorporation of changes, the workflow graph is transformed into a RIH using the *Transform* procedure.

Changes are incorporated into this RIH by the *AddNewPattern* and *ChangeConstraint* procedures in a path-consistent manner and finally the updated RIH is transformed back to its directed graph form using the *InverseTransform* procedure

7.1.1 Contributions

This research work chiefly concentrates in achieving a formal approach which if followed would facilitate composition and maintenance of a workflow in any domain or platform. Thus the main contribution here is a formal architecture for workflow in terms of patterns and temporal constraints amongst them. The originality and innovation in this architecture stems from the contributions that has been achieved and summarized below

Formal specification of workflow - The prime focus of this research work is achieving a formal approach. For this various constructs have been formally defined along with algorithms being devised for processes involved. The definition of WP covers the issue of roles involved in a workflow, temporal constraints amongst the WPs within a workflow and the fact that a WP could form a hierarchy of other patterns. REB construct of TRBAC framework has been successfully used to represent the roles involved in a workflow. For security aspects a trust-based approach has been taken instead of the usual threat-based approach and TE has been defined and used for characterizing SPs. Trust by nature is a user centric proposition. If used properly a trust-based approach makes a security enabled system flexible as against the re-

structive threat-based approach. An SP has been defined in terms of tasks to be performed as against most of previous attempts at defining an SP as a metric to be achieved. This has enabled integrating SPs to WPs resulting in a formal secured version of a WP in the form of a SWP. Realizing the fact that a workflow is a sub-process hierarchy, a RIH has been defined based on the Reference Interval (RI) framework of Allen's IA. RIH formalizes the fact that a set of RIs related by the 13 temporal relations form a hierarchy with a maximal RI as the root that do not refer to any other RI. Definitions of workflow context, security context, WPL, SPL has been induced from the definitions of FCA. For pattern and directed graph, the framework retains the conventional definitions.

Generating and navigating procedures for concept lattices - Exploiting the sub-concept and super-concept relationships amongst the concepts, one can navigate faster through the lattice without visiting each and every concept before the desired concept in the lexicographic order of the lattice. With this understanding, *LatticeGenerator* has been worked out which preserves the sub-concept and super-concept relationships along with the generated concepts.

Generating WPL and SPL- As a formal organization of WPs, WPL has been generated out of Van Der Aalst *et al*'s repository. In the process, WC's has been formalized as characterizing attributes of WPs such that WPs and WC's form the necessary Galois Connection for WPL. Similarly, SPL has been generated as a formal organization of

SPs where the sets of SPs and TEs form the Galois Connection required for SPL. Such a formal approach to organization of patterns have not been attempted before.

Composing and securing a workflow as a directed graph - A workflow could follow different paths of execution based on the constraints being satisfied, thus constituting a directed graph of execution paths. With this understanding, an original approach for composing a workflow as a directed graph of its patterns and constraints amongst them has been given. The approach introduces constraints into the graph in a path consistent manner. Procedure *ComposeWG* has been devised which composes the workflow by introducing and validating the constraints amongst the patterns. *ComposeWG* makes use of two procedures viz *ConstraintAlongWalk* and *ValidateConstraint*. *ConstraintAlongWalk* calculates the transitive constraint along a consistent path between two patterns. *ValidateConstraint* checks for the validity of a constraint between two given patterns.

Opinion has been divided for long on the issue of when to address security aspects of a workflow - during the composition of the workflow or at a later stage as and when required. The approach taken in this research for securing a workflow gives a novel way of addressing the issue in either way. Integrating SPs with SWPs, the given approach generates SWPs which are essentially security enabled WPs. These SWPs could be generated at the time of composition itself and a secured workflow could be composed out of them. Alternatively, after the workflow has been composed, the constituent WPs could be secured as SWPs and

the resultant secured version of the workflow could be validated for consistency by making use of the *ValidateConstraint* procedure

Incorporating changes - Workflow is an evolving system where changes keeps on arising. Robust evolution of a workflow would require accommodating these changes dynamically and in a path consistent manner. Here an innovative approach has been taken for achieving the same. The approach works out three functionalities for incorporating changes - *Transform* procedure that traces a RIH from a workflow graph, *IntroduceNewPattern* and *ChangeConstraint* procedures that incorporates changes into the RIH in a path-consistent manner. *InverseTransform* procedure that transforms back the updated RIH to its directed graph form. In addition to this Table 6.1 has been worked out for consistency check of roles being effected by the changes. It tabulates the feasibility of all possible combination of intervals, periodic expressions and status of a role in the RTs of an REB having the effected role.

7.1.2 Algorithms presented

The approach presented in this thesis has been modularized with algorithms for the various processes and procedures used. The *LatticeGenerator* and *LatticeNavigator* algorithms generate and navigate in the pattern lattices respectively. The complexity of the *LatticeGenerator* algorithm has been discussed. It would take equivalent number of computational steps for generating the whole lattice if the desired pattern is not found and its complexity would be similar to that of the *LatticeGenerator*. One of the implementation

difficulty of these algorithms is if the context at hand is too large, then the number of concepts would be exponential and hence the storage required for the same would be quite huge

Composition of the workflow graph is done by the straightforward algorithm ComposeWG. It works interactively with user inputs. It makes use of the ConstraintAlongWalk for calculating the transitive constraints along various walks between two given patterns. Similarly the ValidateConstraint algorithm is used to validate the consistency of a new constraint in the form of a new pattern being introduced or an existing constraint between two patterns being changed. ValidateConstraint makes use of Allen's procedure to calculate transitive constraints between intervals. ComposeWG would require the Constraint relation to be stored in an auxiliary storage and updated as and when changes are introduced. This updated constraint has to be then cascaded throughout the composition and transform modules and hence synchronization would be one of the implementing issues.

The Transform algorithm traces an RIH from the workflow graph stored in the form of the Constraint relation. It makes a recursive call to the ExploreSubGraph algorithm. ExploreSubGraph explores the subgraph rooted at the currently visited pattern. This algorithm would grow exponentially in the computational steps as the size of the workflow would grow with introduction of new patterns. Again syncing the pair between the Constraint relation and the patterns in the traced RIH each time a new pattern is introduced is one of the implementation issues. The InverseTransform algorithm works exactly in the reverse manner of the Transform algorithm. ~~(skel stub)~~
paus at hand.

7.2 Analysis and Future Work

The scope of this thesis considers ‘pattern’ to be a structurally atomic construct which contains a collection of basic tasks for a recurring problem in a context. However, changes within a ‘pattern’ at task level which may give rise to a new pattern hasn’t been dealt with and could serve as a future direction of work.

The WPL generated includes the four perspectives of Van Der Aalst *et. al.*’s repository of WPs. However, for composition and change incorporation purpose, only the control-flow perspective has been considered. Composition of a workflow from the remaining perspectives could serve as another direction of future work.

The completeness of Van Der Aalst *et. al.*’s repository is not claimed in this research. Therefore explorations whether the repository could be augmented with newer patterns or not could be a potential direction of future work.

There are the following possibilities of completeness and incompleteness of a given context [128]

Case 1: O, A, I are completely known.

Case 2: O is incomplete or too large, A is complete.

Case 3: A is incomplete or too large, O is complete (dual of 2).

Case 4: Neither O nor A is complete.

Case 5: O and A are completely known but relation I is not known.

In the context of a workflow, the set of WPs and WCs will be completely known. The *LatticeGenerator* procedure worked out in this research suffices for the first case. Further research and analysis could be done for procedures of generating concept lattices that would also suffice for the remaining cases. The Titanic algorithm [129] that uses 'support' of an attribute set could be a viable solution for this.

The scope of this research considers changes only in the form of a new pattern being introduced or the value of an existing constraint being changed. Other types of changes in the form of removal of an existing pattern, merging of patterns in sequence etc. need to be handled accordingly.

Formalization of SPs as done here considers that an SP is characterized with pre-conditions and post-conditions though optionally. Thus whenever an SWP is generated from a WP, there remains the issue of how pre-conditions and post-conditions of the SPs in the SWP would be constrained with one another. This hasn't been dealt with in this research and remains as a future direction of work.

The formal definition of TE considers 'Confidence' as a metric that increases temporally in conjugation with its client entity. However, this formalization is done empirically only. Procedurally, how the 'Confidence' measure of a TE would increase over time is yet to be explored and worked out. Another viable direction of future work in this regard would be to find a basis of deciding upon what measure of 'Confidence' could be accepted as a 'Threshold' of a TE.

The overall approach given also serves as a foundation for re-engineering poorly organized workflow of different domains in terms of patterns and con-

straints. Thus an integrated platform wherein different domains could use the proposed approach for re-engineering their workflows could be a future work.

With the advent of Artificial Intelligence, a paradigm shift has occurred in the process of computing. Today computing is more focused towards making a machine learn to perform tasks rather than automating them for specialized tasks as before. In this regard taking forward the formal framework given in this work one can think of achieving a way of inducing newer and context-specific patterns automatically into the pattern lattices. If achieved successfully this would carry forward the research output of this thesis and enhance FCA at the semantic level.

Appendix A

FCA Usage and Tools

Developed

Reference	FCA tools / paradigms used	Framework model / tools developed	Areas
[29]	Concept lattice (of services as objects and words inside service description as attributes)	An approach based on SVM and FCA to automatically classify services to domains and identify concepts in services	CC
[30]	Concept Lattice (for inheritance hierarchy of services - services form objects and keywords in service description form attributes)	Service Explorer - tool (given a set of related services the tool analyze their interface and represents them as a lattice)	CC
[31]	Concept lattice	methodology for the formalization and integration of geographic concepts and relationships encoded in different domain-specific ontologies to reveal their association and interaction	CC
[130]	visualization tools (extracts the implicit data structure from an SME's tacit dimension and characterizes it for external cognition)	FCA is coupled with concept maps for elicitation of Tacit knowledge	CC

[131]	Concept lattices (based on Multiple Classification Ripple down Rule heuristics and folder names of documents retrieved from Web) This lattice is used as an alternate browsing structure for MCRDR	A browsing interface called 'iWeb FCA	CC
[32]	Titanic algorithm for merging contexts of source ontologies of business processes	Formal framework for ontologies	CC
[33]	Titanic algorithm for merging contexts of source ontologies of business processes		CC
[34]	Titanic algorithm for merging contexts of source ontologies of business processes ontology for enhancing FCA applications	conceptual Email Merger -a email management system and 'Courseware watchdog - ontology management system	CC
[35]	Subsumption hierarchy of concept lattice (documents returned by search results are objects and terms in the documents are attributes)	Conceptual and Hierarchical Clustering (CHC) algorithm that results the most relevant concepts for the user query and presents them as a hierarchical structure	CC
[36]	Support function (for selective retrieval of concepts from lattice)	A process of retrieving a sub-lattice based on support function as per the background knowledge of the user on the importance of the concepts in lattice	CC
[38]	formal context (CGs transformed to a formal context)	CGFCA - a tool for transforming a conceptual graph to FORMAL CONTEXT	CC
[39]	Attribute exploitation (for finding missing relations between classes and missing instances)	OntoComp - a plugin for Protege 4	CC
[17]	Inheritance hierarchy of concept lattice (for induction of monotonic linguistic hierarchies)	AOC-poset - A lattice giving the Attribute and Object Concepts	CC DM
[37]	Concept lattice (of documents returned from web queries and terms used in the queries)	A process of query reformulation for refining information retrieval by using concept lattice galois lattice)	CC DM
[132]	Support function (for accelerated computation of concept lattices)	TITANIC algorithm for computing concept lattices	CC DM

[16]	Conceptual clustering, line diagrams, Ganter's Next Closure algorithm	Iceberg concept lattice, "TITANIC" algorithm	CC, DM
[14]	Iceberg concept lattice	framework for rule mining	CC, DM
[5]	inheritance hierarchy, conceptual scaling in concept lattices	Review work of various applications of FCA to linguistics	CC, KR&R.
[12]	Line diagram, concept classification	RDR is combined with FCA to give a knowledge reuse approach.	CC, KR&R.
[27]	Heuristic based lattice pruning (for finding a minimal set of aspects from SMALLTALK code)	"StarBrowser"-A GUI for displaying concepts from lattice	CC, SE
[28]	partition in concept lattice(the lattice has the functions in C code as-objects and their parameters as attributes). A partition gives a module in the code		CC, SE
[13]	Closure of Galois connection (for generating non-redundant association rules)	generic basis for exact association rule and informative basis for approximate association rule	DM
[15]		PASCAL algorithm as an optimization of Apriori algorithm	DM
[40]	Conceptual scales	AnnotationSleuth, a software to experiment with knowledge acquisition using Formal Concept Analysis	KR&R.
[1]	mathematical foundation of FCA		KR&R.
[133]	Galois lattice (as a correspondence between the description lattice (giving the concepts in a domain) and the instance lattice (giving the examples in the domain))	"Gaal" (for GRAPh And Learning) constructs a Galois lattice for any description language provided that the two operations of comparison and generalization are determined for that language	KR&R.
[6]	Conceptual clustering	automatically build a lexicon of subcategorization frames from results of surface parsing. Results derived for the Italian language from several corpora are presented.	KR&R.
[7]	Extends conceptual hierarchies in concept lattices with binary relation among concepts thus giving rise to Relational Concept Analysis (RCA)	A method of classifying meronymy relation using RCA	KR&R.

[8]	subsumption	synthesizes a framework of abstract and partially defined concepts	KR&R
[9]		FCA described in a word box	KR&R
[10]	conceptual hierarchy of Concept Lattice (for depicting relationships among metaphoric classification)	A formal model based on Concept Lattice and Informism that serves as an isomorphic information channel between the contexts of two metaphor domains	KR&R
[11]	Notational definition of concepts	A book that discusses the logic behind information flow of distributed system	KR&R
[18]	Concept Lattice (for clustering execution traces)	'Cable' tool for debugging specifications	SE
[19]	Partial order in concept lattice (execution-time equivalent of control flow implication in program execution traces)	dynamic analog to static control flow relationships in program executions	SE
[20]	Implications derived from concept lattice (of program components as objects and features as attributes) to find urgently required features when program components are executed	Prototype for an Unix environment using GNU tools (gcc nm prof) a concept analysis tool Concepts a graph editor Graphlet and Perl script Presents a case study on two browsers Chimera and Mosaic	SE
[21]	Implications derived from concept lattice (of program components as objects and features as attributes) build on dynamic information of program execution	Prototype for an Unix environment using GNU tools (gcc nm prof) a concept analysis tool Concepts a graph editor Graphlet and Perl script Presents a case study on Xfig tools	SE
[22]	Implications derived from concept lattice (of program components as objects and features as attributes) to find urgently required features when program components are executed		SE
[23]	Hierarchical clustering of concepts (in concept lattices)	A tool for extracting configuration structures from source code	SE
[134]	Sub direct decomposition of concept lattices (used to infer configurations from legacy C++ source code)	'NORA-RECS' tool for defining congruence classes in concept lattices	SE

[24]	Concept Lattice(of code pieces as objects and CPP symbols as attributes) Implication in concept Lattice for finding chain and antichain in lattice lattice decomposition	NORV/RICS - configuration management software	SE
[135]	concept Lattice (for visualizing concepts and concept sub concept relationship among concepts)	An approach to identify groups of functionally related record fields in legacy code using Cluster analysis and concept analysis	SE
[136]	Concept Lattice (for modularizing source code) Implications to transform class hierarchies in the Lattice to a semantically equivalent one showing actual access of the classes in the program	Various processes for modularizing source codes using concept analysis	SE
[137]	Concept Lattice (of abstractions in STL library)	A set of propositions that concept theoretically model the intuitive understanding of good abstractions	SE
[138]	Concept Lattice (with groups of classes as objects and relations among the classes as attributes)	A method of discovering design pattern from Object Oriented code using concept analysis	SE
[139]	Inheritance Galois Lattice (for classifying class protocols in SMALLTALK80)	A formal method for building and maintaining hierarchies of class descriptions	SE
[140]	Line diagram (of a formal context of use cases in OOA/OOD of software) The line diagram assist the choice and definition of objects/classes in domain	A GUI based tool for generating the line diagram	SE
[141]	Line diagram of concept Lattice (where objects and sentences in use cases of natural languages and attributes are phrases in the sentences) Used for reconstructing use cases from different viewpoints for requirement analysis	RECOCASE logic - A viewpoint reconciliation CASE tool	SE
[142]	Concept Lattice (software component as objects and keywords in the components as attributes)	A easy to use incremental component retrieval method based on queries formed from component keywords	SE

[25]	Concept lattice nested line diagrams (concept lattice allows visualizing results returned from graph based queries on information returned by source code analyzer and profilers)	CASS tool (consists of a knowledge base containing software artifacts relationships among artifacts and rules for generating new relationships) These relationships and artifacts are analyzed using FCA	SE
[143]	Heuristic based lattice pruning (for finding a minimal set of aspects from SMALLTALK code)	StarBrowser -A GUI for displaying concepts from lattice	SE
[144]	Concept Lattice (of BPMN flow objects and subsequent concepts in the business domain ontology as attributes) This lattice is used to find cross cutting concerns	A set of tools for cross cutting concern documentation and evolution	SE
[145]	Visualization tools (for navigating and visualizing z-specifications) Concept Lattice(for analyzing and classifying papers on software engineering phases)	SpecTIE-tool for visualizing and navigating Z specifications using FCA	SE
[146]	Concept lattice Implication	A method of analyzing and reengineering class hierarchy	SE
[147]	Concept lattice Implication	A java tool named KABA for analyzing and engineering class hierarchy	SE
[26]		Organize and retrieve artifacts from existing software	SE

Appendix B

Workflow Tools and Deliverables

Reference	Contribution Focus	Tools developed	Area
Review on workflow			
[44]	A formal notation for composition in the form of workflow algebra based on Petri net, A generic definition of workflow view. Introduces the notion of workflow normal forms to remove redundancies and anomalies due to careless composition.	A Petri net based web service composition using workflow view.	business process web service
[48]	Automatic workflow composition from web services using decision theoretic planning based on Markov Decision Process and Bayesian model.	A policy-based model for dynamically composing a workflow from web services.	web service
[49]	Automatic composition of bioinformatics workflow based on EDAM ontology.		life science web service

[53]	Creating workflow for scientific applications (primarily a distributed system)	A approach of composing a workflow for a distributed system where a workflow task is an execution of a program on a target hardware	distributed scientific application
[50]	Workflow composition/completion based on component knowledge base and AI planning	Composition Analysis Tool (CAT) that analyzes workflows and generate error checks and suggestions for users	scientific workflow web service
[45]	An object model for automatic composition of workflow	Z-language based meta model for constrained workflow composition	BPMN semantic web
[54]	OWL based workflow composition of complex scientific process		scientific workflow
[55]	A data-flow based scientific workflow model that separates the interface from the functional body		scientific workflow
[51]	Automatic composition of bioinformatics workflow from different web services repositories	JORCA a desktop client that discovers and invokes web services published in various metadata repositories	bioinformatics web service
[46]	Integration of web services to a e-workflow	The concept of e-workflow which manages e-services and traditional workflow. An algorithm that discovers web services and resolve heterogeneity among their interface and the host workflow (A prototype has been developed)	business process web service
[56]	Workflow composition with temporal role constraints amongst workflow tasks	A software architecture for composition and management of adaptive real time workflows	workflow
[47]	Establishing the application of configuration to automatic and assisted workflow composition	A metamodel for workflow composition based on configuration techniques	web service business process
[52]	Event Calculus (EC)-based planner is used for automatic generation of a workflow from web service	A function ontology that provides semantic function description for Engineering Design Search and Optimization (EDSO) Task ontology that provides semantic descriptions for composite functions	semantic web service

[57]	Deals with dynamic structural changes in WFMS at task level	A graph based model ADBPTflex	Business Process
Review on Workflow Pattern			
[58]	Reduces Bioinformatics workflow development complexity by introducing a set of WPs	Tavaxy, a standalone system for creating and executing workflow based on an extensible set of re-usable WPs. It integrates Taverna and Galaxy workflows	Bioinformatics workflow
[59]	Identifying and analyzing patterns from a Petri-net model	A workflow structure analysis tool called PIPSA Acquisition handling EPC model	Reference workflow model for BPR
[61]	A TGG based model for transforming WPs based on semantic relationships amongst the WPs		Web SOA based system
[62]	The concept and framework of a knowledge workflow	Two extension modules to the WfMC architecture - Pattern-Based Knowledge Flow modeler Intelligent Recommendations Engine A set of knowledge flow patterns	Knowledge flow models
[63]	The concept and framework of a knowledge workflow	A Knowledge Workflow Management System	Knowledge flow models
[64]	Analysis of BPMN notations based on Van Der Aalsts WPs		BPM Van Der Aalsts WPs
[73]	A mapping of Van Der Aalsts control-flow WPs to LOTOS framework		LOTOS Van Der Aalsts WPs
[65]	YAWL UML and BPMN representation of various WP repository (Van Der Aalst, OMG WfMC)		BPM WPs
[66]	Dwyer et al's Property Specification patterns are applied for constructing behavioral properties for workflow (specified in BPMN)	A property specification language PL	BPM
[79]	Increasing expressiveness of Van Der Aalsts representation of WPs by using <i>Cnet</i> formalism		Van Der Aalsts WPs

[60]	Simplifying a Petri net based workflow model	A Pattern Based Process Diagram (PBPD) for identifying patterns from Petri net based workflow A Split-Join Routing Table (SIRT) for analyzing structure related flows in workflow	BPR reference model for workflow
[70]	Evaluation of the support for WPs in Van Der Aalsts repository in ORACLE BPEL PM		Van Der Aalsts WPs Oracle BPEL PM
[80]	Representing workflow resource patterns in Van Der Aalsts repository in Pi-calculus		Van Der Aalsts WPs Pi-calculus
[68]	Increasing expressiveness of Petri-net	Extended Petri-net by introducing D element and C relation	WPs
[67]	A description of the behavioral perspective of workflow	A formal semantics of WPs using Pi-calculus	BPM WP
[72]	Evaluation of the Windows Workflow Foundation on basis of the Van Der Aalsts WPs		Windows Workflow Van Der Aalsts WPs
[71]	Java scripts for implementing Van Der Aalsts WPs in jBOSS JBPM and jPDL		Van Der Aalsts WPs Java
[81]	Use of Petri net synchronic distance for specifying basic WPs	Workflow Management System JBFlo	Van Der Aalsts WPs, Petri net
[82]	Increasing expressiveness of Petri net and formal semantics of complex WPs	Extended Petri net Formal semantics for complex WPs based on Extended Petri net	Petri net Van Der Aalsts WPs
[69]	Reveals ambiguities regarding the partial order of tasks in WP at runtime, entailment specification consolidating related authorization requirements common in workflow domain	A set of task-based entailment constraints revealing potential dependency between conflicting tasks in WPs	Van Der Aalsts WPs workflow access control

Appendix C

Security Patterns Usage

Reference	Contribution Focus	SP organization attempted	Area
[83]	presents an experimental comparison of two techniques for eliciting security requirements - attack trees and misuse cases	none	Security Modeling Techniques
[148]	investigates the qualitative features of the security patterns by providing an evaluation of each pattern based on three main criteria - guidelines regarding how to build secure software exist (Viega and McGraw 2002) main software-hole categories that offer seedbed for possible attacks have been analyzed (Howard and LeBlanc 2002 Viega and McGraw 2002) categories of possible attacks to a system have been identified (Howard and LeBlanc 2002)	none	Information Systems Security
[104]	studies architecture design of application and proposes a catalog of security patterns based on the architectural design of applications	Gives a catalog of architectural patterns on software application layers and components	Application design and architecture

[84]	summarizes how the Method for Performing Diversity and Defense-in-Depth Analysis of Reactor Protection Systems as described in SERC documents	none	System Security Engineering
[85]	A mechanism to precisely specify organizational security patterns for database through out the development life cycle of the database. The methodology uses ADOM and FOOM techniques, and add transformation rules that allow the automatic generation of the desired artifacts based on the specified patterns. Develops a tool called Security Modeling Tool (SMT)	none	Database design security
[86]	The Pattern-based method for Secure Development (PbSD) aims at guiding enforcing and verifying the correct usage of security patterns and utilizing the knowledge encapsulated in these patterns to generate secure applications. It makes use of SMT	none	Database design security
[87]	Investigates application composition guided by competing objectives of performance and security. proposes a heuristic method based on genetic algorithm for service selection based on user preference of performance and security	none	Service oriented computing
[99]	Explores security patterns for different phases of software lifecycle	none a survey report of various catalogs	Software development lifecycle
[88]	Builds a grammar in extended BNF form from Schumacher et al's security patterns to transform project security needs to security requirements	none	Software requirement engineering

[103]	Gives a classification of systems security patterns	Gives a catalog of patterns classified into 8 types	System security
[149]	Transformation of Data Origin Authentication Pattern into Formal Design and Analysis Framework	none, only a single pattern is dealt with	Software security
[105]	Based on Open Group template for pattern classifies security design patterns	Provides a catalog of structural design patterns for secured systems	System design
[150]	Gives a comparative study of selected SPs from existing repositories wherein degrees of fulfillment of security requirements by the patterns is studied	No catalog, instead provides a weighted tabulation of the patterns based on a set of Security Requirements	Application security
[151]	Proposes an approach of hardening aspect oriented code by integrating security in three ways - Code-level hardening (securing code without change in design) Software Process hardening (adding security features into software without changing code) Design-level hardening (re-engineering software to integrate security features that were absent initially)	none	Aspect Oriented Software security hardening
[106]	A comprehensive survey of distributed system SP	A comprehensive catalog of surveyed SPs classification is based on security concern dimensions [Washizaki et al (2009)] and quality characteristics [Laverdiere et al (2006)]	Software security
[100]	Gives a secure software development framework where security patterns are induced at design and development phase	none	Secure development lifecycle

[89]	Gives a mechanism for bridging the gap between enterprise security and software security based on UML extension. Here security aspects are captured in UML based architecture design and subsequently FDAF is applied to the aspect for analysis in the enterprise context it is used.	none	Secured Enterprise software
[101]	Discuss a method of evaluating how the quality of security of a software system is effected by application of SPs. Proposes Misuse patterns that represents general threats to a system that represent low level security which may not be covered by SPs or not considered by the user as important.	none	Security quality of software
[91]	Proposes a Document Security Language (DSL) to secure XML document. The encryption and digital signature of a XML document is stored in a DSL document consisting of six sections - the header, key definition, algorithm definition, security pattern, transformation description, and digital signature.	none	XML security
[118]	Uses the Authenticator pattern for object based distributed system.	none	distributed system security
[90]	Proposes a mechanism Process for Web Service Security (PWSec) that would guide the development of a WS-based security system where each of the development stage is integrated with a complimentary security stage. It has three main stages - risk-based security engineering, pattern-oriented security architecture, and standard centered security design.	No catalog gives a list of new threats of Web service based applications.	web service security

[152]	Proposes a security framework for SOA based on authentication and authorization SPs	none	SOA security
[92]	Gives a three phased development process for secured mobile grid application Planning phase development phase and Maintenance phase	Gives a list of security services then implementing operations and a mapping of security requirements to the security services	Secured Mobile Grid application
[109]	Proposes standardization of automating audit requirements in an organization by giving a list of audit patterns based on security design patterns	Gives a list of AAPs categorized into pattern classes	Business Application Audit
[153]	Defines a security policy pattern for developing applications to securely download code from Internet and execute locally. The pattern can be used either on the client side or the server side. A java code illustration is given.	none	mobile java code
[93]	A methodology for securing system design where security is incorporated at each stage with Human Computer Interface Security patterns	none	Secured system design

[110]	<p>a method to organize and search patterns based on multiple dimensions of classification. Each dimension divides a conceptually continuous space into multiple regions of classification. For each region, patterns are classified by whether or not they play a meaningful role or have value in that region. Multiple orthogonal dimensions are combined to form an n-dimensional space. Patterns occupy regions within that space and can be found by searching and navigating among the regions. Our method builds on the ideas of Personal Construct Theory first described by George Kelly in 1955. In Personal Construct Theory, conceptual dimensions are bi-polar and defined as a continuum between two opposite poles.</p>	<p>Gives a classification of SPs based on multiple dimensions where each dimension cuts across a continuous concern space giving multiple regions.</p>	<p>Secured Concept Grid</p>
[94]	<p>Uses Application-Based Domain Modeling (ADOM) for validating SP used in secure software development. ADM has three layers: Language layer - metamodels and specifications, Domain layer - Building elements of the domain and the relations amongst them, application layer.</p>	<p>none</p>	<p>Secure software layer</p>
[95]	<p>Focuses on development of a security framework for GREDIA project (Grid Enabled To Rich Media Content). The main objective of GREDIA is to develop a platform that would integrate new and existing middleware to support secure business applications.</p>	<p>none</p>	<p>Secured Grid Development</p>

[107]	Represents structural and behavioral information of security patterns using UML notation. It modifies the Design pattern template by Gamma et al with new elements to classify SPs. The newly introduced elements into the design pattern template are - Behavior Constraints, consequences related SPs and supported principles. Makes use of the CheckPoint SP to illustrate the proposed UML based representation.	Gives a new classification of SPs based on the Gamma et al's characteristics - Creational, Structural, Behavioral and Viegas and McGuire's 10 principle.	System security
[102]	Discusses two antipatterns in application security - Perimeter security without proper requirement analysis, consequences of lack of data sensitivity, classification and threat analysis.	none	System security
[154]	Presents a method of automated verification of SP composition by model checking techniques. Formally define the behavioral aspect of security patterns in CCS through their sequence diagrams. We also prove the faithfulness of the transformation from a sequence diagram to its CCS representation.	none	Software security
[98]	Gives a hierarchy of patterns for security of agent based system.	Agency Guard Embassy (specialization) Proxy (specialization) Sandbox (type) - Checkpoint(type) Agent Authenticator (type) - Session (type) Crypto key Generation (type) Crypto key Exchange (type) Access Controller (type)	Agent based system

[111]	proposes seven patterns for application security	<p>Single Access Point: Providing a security module and a way to log into the system</p> <p>Check Point: Organizing security checks and their repercussions</p> <p>Roles: Organizing users with similar security privileges</p> <p>Session: Localizing global information in a multi-user environment</p> <p>Full view with errors: Provide a full view to users showing exceptions when needed</p> <p>Limited view: Allowing users to only see what they have access to</p> <p>Secured Access Layer: Integrating application security with low level security</p>	Application security
[112]	proposes 26 SPs divided into structural and procedural patterns	Gives a catalog of 26 SPs for web application development	Web application security
[113]	Gives a set of SPs to give security enhanced sendmail in the form of Qmail		Qmail security
[108]	Gives a list of 14 SPs classified according to the CIA model application context Microsoft Classification model and the STRIDE model	Four classification of the 14 SP patterns listed	Security patterns
[96]	Defines two types of aspects for security - Generic aspect like an attack pattern which is platform independent and a context-specific pattern which is an instantiation of generic aspect security treated model of the system is composed with context-specific attack aspects to give the security-treated misuse model. This model is analyzed to ensure that the given attack is mitigated	none	Aspect Oriented Application Design

[97]	Considers SPs as design patterns Evaluates 22 of Schumacher's SPs for learning purpose	none	Design patterns
[155]	a model-driven transformation framework that generates security configurations out of annotated business process models	none	System security

Appendix D

WP and WC Enumeration

Workflow Patterns		
WP 1 : Name: <i>Sequence</i> perspective: <i>Control flow</i> <i>V_{index}: C1</i>	WP 2 : Name: <i>Parallel split</i> perspective: <i>Control flow</i> <i>V_{index}: C2</i>	WP 3 : Name: <i>synchronization</i> perspective: <i>Control flow</i> <i>V_{index}: C3</i>
WP 4 : Name: <i>Multi-choise</i> perspective: <i>Control flow</i> <i>V_{index}: C6</i>	WP 5 : Name: <i>Structured Synchronizing Merge</i> perspective: <i>Control flow</i> <i>V_{index}: C7</i>	WP 6 : Name: <i>Structured Discriminator</i> perspective: <i>Control flow</i> <i>V_{index}: C9</i>

<p>WP 7 : Name: <i>Arbitrary Cycle</i> perspective: <i>Control flow</i> <i>V_{index}: C10</i></p>	<p>WP 8 : Name: <i>Multiple Instance Without Synchronization</i> perspective: <i>Control flow</i> <i>V_{index}: C12</i></p>	<p>WP 9 : Name: <i>Task Data</i> perspective: <i>Data</i> <i>V_{index}: D1</i></p>
<p>WP 10 : Name: <i>Task To Data</i> perspective: <i>Data</i> <i>V_{index}: D8</i></p>	<p>WP 11 : Name: <i>Task Precondition Data Existence</i> perspective: <i>Data</i> <i>V_{index}: D33</i></p>	<p>WP 12 : Name: <i>Task Precondition Data Value</i> perspective: <i>Data</i> <i>V_{index}: D34</i></p>
<p>WP 13 : Name: <i>Direct Alloc</i> perspective: <i>Resource</i> <i>V_{index}: R1</i></p>	<p>WP 14 : Name: <i>Distribution by Offer- Single Resource</i> perspective: <i>Resource</i> <i>V_{index}: R12</i></p>	
Workflow Concerns		
<p>WC 1 : Perspective: <i>Control flow</i> Concern: <i>Initialization</i> Description: <i>Relates to the initialization/pre-condition of a task before it's activation/execution</i></p>	<p>WC 2 : Perspective: <i>Control flow</i> Concern: <i>Finalization</i> Description: <i>Relates to the finalization of state variables/post-condition of a task</i></p>	<p>WC 3 : Perspective: <i>Control flow, Data</i> Concern: <i>Synchronization</i> Description: <i>Relates to the messaging scheme amongst simultaneous execution of multiple tasks / multiple instances of the same task</i></p>

<p>WC 4 : Perspective: <i>Control flow</i> Concern: <i>Execution order</i> Description: <i>Relates to the bookmarking archiving of state of a task etc. required in branched execution</i></p>	<p>WC 5 : Perspective: <i>Control flow</i> Concern: <i>Iteration</i> Description: <i>Relates to iterative execution of a set of tasks</i></p>	<p>WC 6 : Perspective: <i>Control flow</i> Concern: <i>Decision</i> Description: <i>Relates to conditional execution of a set of tasks</i></p>
<p>WC 7 : Perspective: <i>Control flow Data</i> Concern: <i>Visibility</i> Description: <i>Relates to the accessibility of data elements in a workflow</i></p>	<p>WC 8 : Perspective: <i>Control flow Data</i> Concern: <i>Interaction</i> Description: <i>Relates to the manner of communication of data amongst active elements of workflow</i></p>	<p>WC 9 : Perspective: <i>Control flow Data</i> Concern: <i>Transfer</i> Description: <i>Relates to the way in which data elements are actually transferred between workflow components</i></p>
<p>WC 10 : Perspective: <i>Data</i> Concern: <i>Routing</i> Description: <i>Relates to the manner in which data elements can influence the operation of other perspective of the workflow</i></p>	<p>WC 11 : Perspective: <i>Control flow, Data, Resource</i> Concern: <i>Creation</i> Description: <i>Relates to limitations on the manner in which a work item may be executed. They serve to restrict the range of resources that can undertake work items that correspond to the task. They also influence the mapping of a work item with competent resources</i></p>	<p>WC 12 : Perspective: <i>Resource</i> Concern: <i>Push allocation</i> Description: <i>Relates to the manner in which newly created work items are allocated to resources by the workflow engine</i></p>

<p>WC 13 : Perspective: <i>Resource</i> Concern: <i>Visibility</i> Description: <i>Relates to the accessibility of an work item to various resources</i></p>	<p>WC 14 : Perspective: <i>Resource</i> Concern: <i>Pull allocation</i> Description: <i>Relates to the manner in which newly created work items are advertised to resources. Here the commitment for execution of the work item comes from the resource rather than being allocated by the workflow engine</i></p>	<p>WC 15 : Perspective: <i>Resource</i> Concern: <i>Detour</i> Description: <i>Relates to the manner in which the resource executing a work item may be changed during execution</i></p>
<p>WC 16 : Perspective: <i>Resource</i> Concern: <i>Auto start</i> Description: <i>Relates to the manner in which an instance of a work item may start automatically. It deals with the role the executing resource needs to play when a work item auto starts itself.</i></p>	<p>WC 17 : Perspective: <i>Resource</i> Concern: <i>Multiple execution of work items by a resource</i> Description: <i>Relates to a many to many correspondence between resources and work items</i></p>	

Appendix E

SP and TE Enumeration

SP enumeration		
SP 1 : NAME: <i>AUTHENTICATOR(ATH)</i> TASK: <i>Accept and verify credentials</i> EXHIBIT: <i>Result of verification</i>	SP 2 : NAME: <i>AUTHORIZER(ATR)</i> TASK: <i>Grant privilege to user on resource</i> EXHIBIT: <i>User gains access to resource</i>	SP 3 : NAME: <i>CHECKPOINT (CP)</i> TASK: <i>Insert a flag with time stamp and/or stateinfo in a system under execution. SP 6 may be applied on this flag in the future</i> EXHIBIT: <i>A new secured version of the system is achieved</i>

<p>SP 4 : NAME: DEFENSE IN DEPTH (DID) TASK: Insert security check at each layer of the application SPs 1 and 2 are applied here EXHIBIT: Access to next layer</p>	<p>SP 5 : NAME: USER DEFINED EXCEPTION (UDE) TASK: Encapsulate unsafe runtime exceptions with user defined exceptions that is safe by design EXHIBIT: Command to user regarding exception handling or exception handling routine</p>	<p>SP 6 : NAME: ROLLBACK(RB) TASK: Restore system to state before a checkpoint flag EXHIBIT: Previous working version of the system</p>
<p>SP 7 : NAME: PROOF(PF) TASK: Justify the rationality of of an object to be rationalized based on some base theorem EXHIBIT: New hypothesis for the object to be rationalized</p>	<p>SP 8 : NAME: SECURE-PREFORK(SPF) TASK: Recreate pool of pre-forked process in the pool EXHIBIT: New pool of process</p>	<p>SP 9 : NAME: ESTABLISH_CONNECTION(ECON) TASK: Create dedicated end-to-end channel between communicating parties EXHIBIT: Establish tunnel through which the two parties can communicate</p>
<p>SP 10 : NAME: SYNCHRONIZATION(SYN) TASK: Schedule or protocol for a global resource EXHIBIT: Multiple user conflict resolved</p>	<p>SP 11 : NAME: GARBAGE COLLECTION(GC) TASK: De allocate expired objects / processes and resolve dangling references EXHIBIT: Freed application buffer</p>	<p>SP 12 : NAME: DYNAMIC MEMORY ALLOCATOR(DMA) TASK: Allocate memory to object reference from buffer EXHIBIT: Access to created runtime object</p>

<p>SP 13 : NAME: <i>NORMALIZATION(NRM)</i> TASK: <i>Remove redundancy of attributes and resolve dependency between objects</i> EXHIBIT: <i>Normal form of objects with enforced integrity</i></p>	<p>SP 14 : NAME: <i>REFRESH(RFR)</i> TASK: <i>Reload buffer with new content/ Recreate runtime objects after lifetime expiry</i> EXHIBIT: <i>Newly created objects updated buffer content</i></p>	<p>SP 15 : NAME: <i>INDUCTION(IND)</i> TASK: <i>Induce objects based on existing objects and dependencies</i> EXHIBIT: <i>Completeness achieved</i></p>
<p>SP 16 : NAME: <i>LOGIC(LG)</i> TASK: <i>Formalize process description</i> EXHIBIT: <i>Code / Pseudocode</i></p>		
<p>WC Enumeration</p>		
<p>Identity(ID): Unique identifier of a resource Access Policy(AP): Policy defining user-resource access details in a context Retention(RT): Property of holding state information Authority(ATH): Power of execution or possession of resource Authenticity(ATC): Property of genuineness or legitimacy Memory buffer size(MBS) Size of buffer available Integrity(INT): Reliability of a resource Confidentiality(CON) Property of discretion or privacy Privilege(PRV): Freedom or opportunity of use Formality(FRM): Property of having a generic procedure or rule of execution Completeness(COM): Property of totality or closeness Confirmity(CNF): Property of being substantiated with proof or believe Consistency(CNS): Property of having similar behavior over a period of time Persistence(PRS): Property of holding state info permanently Uniqueness(UNQ): Property of having single representation</p>		

Appendix F

Patterns borrowed from Van Der Aalst's repository

F.1 Pattern name - Sequence

- **Description** - A task in a process is enabled after the completion of a preceding task in the same process
- **Synonyms** - Sequential routing, serial routing
- **Examples** -
 - The verify-account task executes after the credit card details have been captured
 - The codacil-signature task follows the contract-signature task
 - A receipt is printed after the train ticket is issued
- **Motivation** - The Sequence pattern serves as the fundamental building block for processes. It is used to construct a series of consecutive tasks which execute in turn one after the other. Two tasks form part of a Sequence if there is a control-flow edge from one of them to the next which has no guards or conditions associated with it.
- **Context** - There is one context condition associated with this pattern: an instance of the Sequence pattern cannot be started again until it has completed execution of the preceding thread of control (i.e. all places such as p1 in the Sequence must be safe).

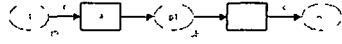


Figure F 1 Sequence Pattern

Pattern name - Exclusive Choice

- **Description** - The divergence of a branch into two or more branches. When the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on the outcome of a logical expression associated with the branch.
- **Synonyms** - XOR-split, exclusive OR-split, conditional routing, switch, decision, case statement.
- **Examples** -

- After the review election activity is complete, either the declare results or the recount votes activity is undertaken.

- **Motivation**- The Exclusive Choice pattern allows the thread of control to be directed to a specific activity depending on the outcome of a preceding activity, the values of elements of specific data elements in the workflow or the results of a user decision. The routing decision is made dynamically, allowing it to be deferred to the latest possible moment at runtime.
- **Context** -

The behaviour of the Exclusive Choice pattern is illustrated by the CPN model in Figure F 2. Depending on the results of the condition expression, the thread of control is either routed to activity B or C. There are two context conditions associated with this pattern: (1) the information required to calculate the logical conditions on each of the outgoing branches must be available at runtime at the point at which the choice construct is reached in the process and (2) the condition associated with precisely one outgoing branch of the exclusive choice construct must evaluate to true.

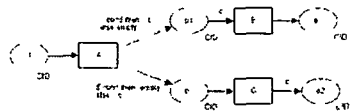


Figure F 2 Exclusive Choice Pattern

Pattern name - Parallel split

- **Description** - The divergence of a branch into two or more parallel branches each of which execute concurrently
- **Synonyms** - AND-split parallel routing parallel split fork
- **Examples** -
 - When an intrusion alarm is received trigger the despatch patrol activity and the inform police activity immediately
 - Once the customer has paid for the goods issue a receipt and pack them for despatch
- **Motivation**- The Parallel Split pattern allows a single thread of execution to be split into two or more branches which can execute activities concurrently. These branches may or may not be re-synchronized at some future time
- **Context** - Figure F 3 illustrates the implementation of the Parallel Split. After activity A has completed two distinct threads of execution are initiated and activities B and C can proceed concurrently

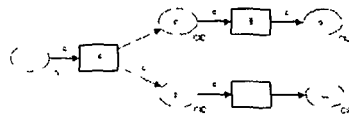


Figure F 3 Parallel Split Pattern

F.2 Pattern name - Structured Loop

- **Description** - The ability to execute an activity or sub-process repeatedly. The loop has either a pre-test or post-test condition associated with it that is either evaluated at the beginning or end of the loop to determine whether it should continue. The looping structure has a single entry and exit point
- **Synonyms** -
- **Examples** -
 - While the machine still has fuel remaining continue with the production process
 - Continue processing photographs from the Im until all of them have been printed
 - Repeat the select player activity until the entire team has been selected

- **Motivation-** There are two general forms of this pattern - the while loop which equates to the classic while - do pre-test loop construct used in programming languages and the repeat loop which equates to the repeat - until post-test loop construct

The while loop allows for the repeated sequential execution of a specified activity or a sub-process zero or more times providing a nominated condition evaluates to true. The pre-test condition is evaluated before the 1st iteration of the loop and is re-evaluated before each subsequent iteration. Once the pre-test condition evaluates to false, the thread of control passes to the activity immediately following the loop. The while loop structure ensures that each of the activities embodied within it are executed the same number of times.

The repeat loop allows for the execution of an activity or sub-process one or more times, continuing with execution until a nominated condition evaluates to true. The post-test condition is evaluated after the 1st iteration of the loop and is re-evaluated after each subsequent iteration. Once the post-test condition evaluates to true, the thread of control passes to the activity immediately following the loop. The repeat loop structure ensures that each of the activities embodied within it are executed the same number of times.

- **Context -** As indicated above, there are two variants of this pattern - the while loop illustrated in Figure F 4 and the repeat loop shown in Figure F 5. In both cases, activity B is executed repeatedly.

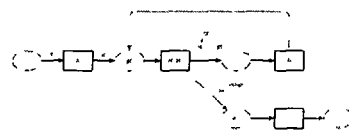


Figure F 4 Structured Loop (while) Pattern

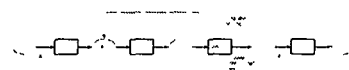


Figure F 5 Structured Loop (repeat) Pattern

Bibliography

- [1] Ganter, B & Wille, R. Formal concept analysis-mathematical foundation *Berlin Springer-1999* (1999)
- [2] Bertino, E Bonatti, P A & Feriani, E TRBAC A temporal role-based access control model *ACM Transactions on Information and System Security* **4**, 191–233 (2001) URL [http //doi acm org/10 1145/501978 501979](http://doi.acm.org/10.1145/501978.501979)
- [3] Allen J F Maintaining knowledge about temporal intervals *Communications of the ACM* **26** 832–843 (1983) URL [http //doi acm org/10 1145/182 358434](http://doi.acm.org/10.1145/182.358434)
- [4] Ganter & Stumme *Formal Concept Analysis Methods and Applications in Computer Science Adapted by G Stumme* (Springer-Heidelberg, 2003)
- [5] Priss, U Linguistic applications of formal concept analysis In Ganter, B, Stumme, G & Wille, R. (eds) *Formal Concept Analysis* vol 3626 of *Lecture Notes in Computer Science*, 149–160 (Springer Berlin Heidelberg 2005)

- [6] Basili, R., Pazienza, M. T. & Vindigni, M. Corpus-driven unsupervised learning of verb subcategorization frames. In *AI*IA '97: Proceedings of the 5th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence*, 159–170 (Springer-Verlag, London, UK, 1997).
- [7] Priss, U. Classification of meronymy by methods of relational concept analysis. In *proceedings of the 1996 Midwest Artificial Intelligence Conference* (1996).
- [8] Woods, W. A. Understanding subsumption and taxonomy: framework for progress. *Principles of Semanteic Network: Explorations in the Representation of Knowledge* 45–94 (1991).
- [9] Wille, R. & Kipke. Formale begriffsanalyse erlautert an einem wortfeld. In *LDV-Forum* 5, 31–36 (1987).
- [10] Old, L. J. & Priss, U. Metaphor and information flow. In *proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference*, 99–104 (2001).
- [11] Barwise, J. & Seligmann, J. *Infomation Flow: The logic of Distributed Systems* (Cambridge University Press, 1997).
- [12] Richards, D. & Compton, P. Combining formal concept analysis and ripple down rules to support the reuse of knowledge. In *proceedings of the Ninth International Conference on Software Engineering and Knowledge Engineering SEKE' 97, Madrid, Spain*, 177–184 (1997).

- [13] Bastide Y, Pasquier, N, Taouil R, Stumme G & Lakhal, L Mining minimal non-redundant association rules using frequent closed itemsets. In *Proceedings of the First International Conference on Computational Logic*, CL '00, 972–986 (Springer-Verlag, London, UK, UK, 2000) URL http://dl.acm.org/citation.cfm?id=647482_728293
- [14] Lakhal, L & Stumme, G Efficient mining of association rules based on formal concept analysis. In Gantner, B, Stumme, G & Wille, R (eds) *Formal Concept Analysis*, vol. 3626 of *Lecture Notes in Computer Science*, 180–195 (Springer Berlin Heidelberg, 2005)
- [15] Bastide, Y, Taouil, R, Pasquier, N, Stumme, G & Lakhal, L Mining frequent patterns with counting inference. *SIGKDD Explor. Newsl.* **2**, 66–75 (2000) URL <http://doi.acm.org/10.1145/380995.381017>
- [16] Stumme, G, Taouil, R, Bastide, Y, Pasquier, N & Lakhal, L Computing iceberg concept lattices with titanic. *Data Knowl. Eng.* **42**, 189–222 (2002) URL [http://dx.doi.org/10.1016/S0169-023X\(02\)00057-5](http://dx.doi.org/10.1016/S0169-023X(02)00057-5)
- [17] Osswald, R & Petersen, W Induction of classification from linguistic data. In *proceedings of the ECAI- Workshop on advances in Formal Concept Analysis for Knowledge Discovery in Databases* (Lyon, 2002)
- [18] Ammons, G, Mandel, D, Bodík, R & Larus, J R Debugging temporal specifications with concept analysis. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and im-*

plementation, PLDI '03, 182–195 (ACM, New York NY, USA, 2003)
URL <http://doi.acm.org/10.1145/781131.781152>

- [19] Bell T The concept of dynamic analysis In *Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-7, 216–234 (Springer-Verlag, London, UK, UK, 1999) URL <http://dx.doi.org/10.1145/318773.318944>
- [20] Eisenbaith, T , Koschke, R & Simon, D Aiding program comprehension by static and dynamic feature analysis In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 01)* ICSM '01, 602–611 (IEEE Computer Society, Washington, DC, USA, 2001) URL <http://dx.doi.org/10.1109/ICSM.2001.972777>
- [21] Eisenbaith, T , Koschke, R & Daniel, S Feature-driven program understanding using concept analysis of execution traces In *Proceedings of the 9th International Workshop on Program Comprehension IWPC '01*, 300–309 (IEEE Computer Society, Washington, DC USA, 2001) URL <http://dl.acm.org/citation.cfm?id=876902.881278>
- [22] Eisenbaith, T , Koschke, R & Simon, D Locating features in source code *IEEE Trans Softw Eng* **29**, 210–224 (2003) URL <http://dx.doi.org/10.1109/TSE.2003.1183929>
- [23] Krone M & Snelling G On the inference of configuration structures from source code In *Proceedings of the 16th international conference on Software engineering*, ICSE '94, 49–57 (IEEE

Computer Society Press, Los Alamitos, CA, USA, 1994) URL <http://dl.acm.org/citation.cfm?id=257734.257742>

- [24] Snelling, G Reengineering of configurations based on mathematical concept analysis *ACM Trans Softw Eng Methodol* 5, 146–189 (1996) URL <http://doi.acm.org/10.1145/227607.227613>
- [25] Cole & Tilley Conceptual analysis of software structure In *Fifteenth International Conference on Software Engineering and Knowledge Engineering, SEKE 03*, 726–733 (USA, Knowledge Systems Institute, 2003)
- [26] Godin, R , Mineau, G W , Missaoui, R , German, M & Faraj, N Applying concept formation methods to software reuse *International Journal of Software Engineering and Knowledge Engineering* 5, 119–142 (1995)
- [27] Mens, K & Touwe, T Reverse engineering aspectual views using formal concept analysis *Position paper in Workshop on Object Oriented Reengineering in ECOOP(2004), Oslo* (2004)
- [28] Siff, M & Reff, T Identifying modules using concept analysis *IEEE Transactions on Software Engineering* 25, 749–768 (1999)
- [29] Bruno, M , Canfora, G , Penta, M D & Scognamiglio, R An approach to support web service classification and annotation In *In Proc of IEEE International Conference on e-Technology e-Commerce and e-Service*, 138–143 (IEEE Press 2005)

- [30] Aversano, L Bruno, M , Penta, M D , Falanga, A & Scognamiglio, R
Visualizing the evolution of web services using formal concept analysis
In *proceedings of the Eighth International Workshop on Principles of
Software Evolution , IEEE Computer Society* 57–60 (2005)
- [31] Kokla, M & Kavouas, M Concept lattices as a formal method for the
integration of geospatial ontologies In *proceedings of EuroConference
on Ontology and Epistemology for Spatial Data Standards* (2000)
- [32] Stumme, G Using ontologies and formal concept analysis for orga-
nizing business knowledge In *proceedings of Referenzmodellierung*,
163–174 (Physica, 2002)
- [33] Stumme, G & Maedche A FCA-Merge Bottom-up merging of on-
tologies In *proceedings of IJCAI*, 225–234 (2001)
- [34] Hotho, A , Stumme, G & Tane, J Conceptual knowledge processing
with formal concept analysis and ontologies In *proceedings of Sec-
ond International Conference on Formal Concept Analysis in Sydney,
Australia*, 189–207 (2004)
- [35] Zhang, Y & Feng, B Clustering search results based on formal concept
analysis *Information Technology Journal* 746–753 (2008)
- [36] Belohlavek, R & Vychodil, V Background knowledge in for-
mal concept analysis constraints via closure operators In *Pro-
ceedings of the 2010 ACM Symposium on Applied Computing*,
SAC 10 1113–1114 (ACM New York NY USA, 2010) URL
[http //doi acm org/10 1145/1774088 1774322](http://doi.acm.org/10.1145/1774088.1774322)

- [37] Qadi, A. E., Aboutajdine, D. & Ennouay, Y. Formal concept analysis for information retrieval. *International Journal of Computer Science and Information Security* **7**, 119–125 (2010)
- [38] Andrews, S. & Polovina, S. A mapping from conceptual graphs to formal concept analysis. In Andrews, S., Polovina, S., Hill, R. & Akhgar, B. (eds.) *Conceptual Structures for Discovering Knowledge*, vol. 6828 of *Lecture Notes in Computer Science*, 63–76 (Springer Berlin Heidelberg, 2011)
- [39] Sertkaya, B. Talk on 'ontology completion with formal concept analysis' in sap research lab, germany. *Dept. of Theoretical Computer Science TU Dresden* (2009)
- [40] Richards, D. & Kang, B. H. Knowledge acquisition: Approaches, algorithms and applications. pacific rim knowledge acquisition workshop, plaw 2008, hanoi, vietnam, december 15-16, 2008, revised selected papers. In *PKAW*, vol. 5465 of *Lecture Notes in Computer Science* (Springer, 2009)
- [41] Kuznetsov, S. & Obiedkov, S. Comparing performance of algorithms generating formal concept lattices. *Journal of experimental and Theoretical Artificial Intelligence* **14**, 189–216 (2002)
- [42] Ganter, B. Two basic algorithms in concept analysis. In Kwuida, L. & Sertkaya, B. (eds.) *Formal Concept Analysis*, vol. 5986 of *Lecture Notes in Computer Science*, 312–340 (Springer Berlin Heidelberg, 2010)

- [43] Obiedkov, S. & Duquenne, V. Attribute-incremental construction of the canonical implication basis. *Annals of Mathematics and Artificial Intelligence* **49**, 77–99 (2007).
- [44] Pankratius, V. & Stucky, W. A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets. In *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling*, vol. 43 of *APCCM '05*, 79–88 (Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 2005). URL <http://dl.acm.org/citation.cfm?id=1082276.1082286>.
- [45] Albert, P., Henocque, L. & Kleiner, M. A constrained object model for configuration based workflow composition. In *Proceedings of the Third international conference on Business Process Management, BPM'05*, 102–115 (Springer-Verlag, Berlin, Heidelberg, 2006).
- [46] Cardoso, J. & Sheth, A. Semantic e-workflow composition. *Journal of Intelligent Information Systems* **21**, 191–225 (2003). URL <http://portal.acm.org/citation.cfm?id=940053>.
- [47] Albert, P., Henocque, L. & Kleiner, M. Configuration-based workflow composition. *IEEE International Conference on Web Services ICWS05* 285–292 (2005). URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1530808>.

- [48] Doshi P, Goodwin R, Akkiraju R & Veina K. Dynamic workflow composition using markov decision processes. In *Proceedings IEEE International Conference on Web Services*, vol. 2, 576–582 (Ieee, 2004)
- [49] Lamprecht A-L, Naujokat S, Steffen B & Margaria T. Constraint-guided workflow composition based on the edam ontology. In *Proceedings of the 3rd International Workshop on Semantic Web Applications and Tools for the Life Sciences Berlin, Germany December 10* (2010) URL <http://arxiv.org/abs/1012.1640>
- [50] Kim, J, Spiraragen, M & Gil, Y. An intelligent assistant for interactive workflow composition. In *Proceedings of the 9th international conference on Intelligent user interfaces, IUI '04*, 125–131 (ACM, New York, NY USA, 2004)
- [51] Karlsson J, Martín-Requena, V, Rios, J & Tielles, O. *Workflow Composition and Enactment Using jORCA*, vol. 6415, 328–339 (Springer Berlin Heidelberg, 2010) URL <http://www.springerlink.com/content/2n5u695r1523kp73/>
- [52] Chen, L & Yang, X. Applying AI planning to semantic web services for workflow generation. In *First International Conference on Semantics Knowledge and Grid, SKG 05*, 65–67 (2005)
- [53] Nguyen, B M, Tian, V & Hluchy, L. Programmable workflow composition. In *The 2nd International Conference on Next Generation Information Technology (ICNIT)*, 86–89 (2011)

- [54] Gil, Y. Workflow composition: Semantic representations for flexible automation. *Workflows for eScience* 1–15 (2007)
- [55] Fei, X. & Lu, S. A dataflow-based scientific workflow composition framework. *IEEE Transactions on Services Computing* **99**, 1–14 (2010)
- [56] Shafiq, B., Samuel, A. & Ghafoor, H. A GTRBAC based system for dynamic workflow composition and management. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC 2005*, 284–290 (2005)
- [57] Reichert, M. & Dadam, P. Adept flex - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems* **10**, 93–129 (1998)
- [58] Abouelhoda, M., Issa, S. A. & Ghanem, M. Taxaxy: Integrating taxena and galaxy workflows with cloud computing support. *Open access journal 'BMC Bioinformatics'* **13** (2012). URL <http://www.ncbi.nlm.nih.gov/pubmed/22559942>
- [59] Cheng, H. J., Ou-Yang, C. & Juan, Y. A pattern based approach to workflow structure analysis. In *IEEE 18th International Conference on Industrial Engineering and Engineering Management (IEEM)*, vol. 3, 2145–2149 (2011)
- [60] Ou-Yang, C. & Cheng, H. J. A pattern based approach for structural analysis of workflow models. *International Journal of Innovative Computing, Information and Control* **8**, 5217–5336 (2012)

- [61] Lohmann, C., Greenyer, J. & Jiang, J. Applying triple graph grammars for pattern-based workflow model transformations. *Journal of Object Technology, Special Issue: TOOLS EUROPE 2007* **6**, 253–273 (2007).
- [62] Surendra Sarnikar, J. L. Z. Pattern-based knowledge workflow automation: concepts and issues. *Information Systems and e-Business Management* **385–402** (2008).
- [63] Sarnikar, S. *Automating knowledge flows by extending conventional information retrieval and workflow technologies*. Ph.D. thesis, Tucson, AZ, USA (2007). AAI3243880.
- [64] Wohed, P., Aalst, W. M. P. V. D. & Dumas, M. Pattern-based analysis of BPMN- an extensive evaluation of the control-flow, the data and the resource perspectives. *Technology* **14**, 781–787 (2005).
- [65] Fortis, A. & Fortis, F. Workflow patterns in process modeling. *Arxiv preprint arXiv09030053* **14** (2009). URL <http://arxiv.org/abs/0903.0053>.
- [66] Wong, P. Y. H. & Gibbons, J. Property specifications for workflow modelling. *Science of Computer Programming* **76**, 942–967 (2011).
- [67] Puhlmann, F. & Weske, M. Using the pi-calculus for formalizing workflow patterns. *Education* **3649**, 153–168 (2005).
- [68] Zhang, L., Yao, S. & Li, J. Formalizing workflow patterns with extended petri-net. *Sixth International Conference on Natural Computation ICNC* **6**, 3164–3168 (2010).

- [69] Wolter, C, Schaad, A & Menel, C Task-based entailment constraints for basic workflow patterns In *Proceedings of the 13th ACM symposium on Access control models and technologies SACMAT 08* 51-60 (ACM Press, 2008) URL http://portal.acm.org/citation.cfm?id=1377836_1377844
- [70] Mulayi, N A Pattern-based evaluation of oracle-bpel (v 10 1 2) Tech Rep, Department of Technology Management, Eindhoven University of Technology The Netherlands (2005)
- [71] Peng, L & Zhou, B Research on workflow patterns based on jBPM and jPDL In *Proceedings of the 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application - Volume 02, PACIIA '08* 838-843 (IEEE Computer Society, Washington, DC, USA, 2008)
- [72] Zapletal M Van Der Aalst W M P Russell, N, Liegl P & Weithner, H Pattern-based analysis of windows workflow *Eindhoven University of Technology Tech Rep CSReport 09-07* (2009)
- [73] Takecian, P, Ferreira, J, Malkowski, S & Pu, C Using lotos for rigorous specifications of workflow patterns In *6th International Conference on Collaborative Computing Networking, Applications and Workshoping (CollaborateCom)*, 1-7 (2010)
- [74] Russell, N Hofstede A H M T & Mulayi, N Workflow control-flow patterns A revised view Tech Rep Queensland University of Technology (2006)

- [75] Russell N, Hofstede, A H M T & Edmond D Workflow data patterns Tech Rep Queensland University of Technology (2004)
- [76] Russell, N, Terhofstede, A M, Edmond, D & V, W P Workflow data patterns Identification, representation and tool support In *Proceedings of the 25th International Conference on Conceptual Modeling (ER 2005)* 353–368 (Springer 2005)
- [77] Russell, N, Hofstede, A H M T & Edmond, D Workflow resource patterns Tech Rep In the 17th Conference on Advanced Information Systems Engineering (CAISE05) (2004)
- [78] Russell, N van der Aalst & ter Hofstede, W Exception handling patterns in process-aware information systems Tech Rep BPMcenter.org (2006)
- [79] Zhang L & Yao, S Using the c net for formalizing workflow patterns In *Second International Conference on Information Technology and Computer Science*, 102–105 (IEEE, 2010) URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5557319>
- [80] Xue, G, Lu J Gong N & Yao, S Investigating workflow resource patterns in term of pi-calculus In *12th International Conference on Computer Supported Cooperative Work in Design 2008 CSCWD*, 630–635 (2008)
- [81] Zhao W Huang Y & Yuan C Synchronic distance based workflow logic specification In *High Performance Computing and Commu-*

ications 2008 HPCC 08 10th IEEE International Conference on,
819–824 (2008)

- [82] Zhang L Research on workflow patterns based on petri nets
2006 IEEE Conference on Robotics Automation and Mechatronics 1–6
(2006) URL <http://dx.doi.org/10.1109/RAMECH.2006.252634>
- [83] Opdahl, A L & Sindre G Experimental comparison of attack
trees and misuse cases for security threat identification *In-*
formation and Software Technology **51**, 916–932 (2009) URL
<http://linkinghub.elsevier.com/retrieve/pii/S0950584908000773>
- [84] Bayuk J L, Horowitz, B M & Jones, R A Security via
related disciplines *Procedia CS* **8**, 338–344 (2012) URL
<http://dblp.uni-trier.de/db/journals/procedia/procedia8.html>
- [85] Abramov J, Anson, O, Dahan, M, Shoval, P & Stum, A A method-
ology for integrating access control policies within database develop-
ment *Computers Security* **31** 299–314 (2012)
- [86] Abramov J, Stum A & Shoval P Evaluation of the pattern-based
method for secure development (pbsd) a controlled experiment *In-*
formation and Software Technology **54**, 1029–1043 (2012)
- [87] Zo H, Nazareth D L & Jain, H K Security and perfor-
mance in service-oriented applications Trading off competing ob-
jectives *Decision Support Systems* **50**, 336–346 (2010) URL
<http://dl.acm.org/citation.cfm?id=1879269>

- [88] Supapoin, K , Prompoon N & Rojkangsadan T An approach Constructing the grammar from security pattern *Proc of International Joint Conference on Computer Science and Software Engineering JC-SSE2007* (2007)
- [89] Dai L & Cooper, K Using FDAF to bridge the gap between enterprise and software architectures for security *Science of Computer Programming* **66**, 87-102 (2007) URL <http://dl.acm.org/citation.cfm?id=1234406> 1234429
- [90] Gutierrez C Rosado D G & Fernandez-Medina E The practical application of a process for eliciting and designing security in web service systems *Information and Software Technology* **51**, 1712-1738 (2009)
- [91] Hwang, G -H & Chang, T -k An operational model and language support for securing xml documents *Computers Security* **23** 498-529 (2004)
- [92] Rosado D G Fernandez-Medina E Lopez J & Plattini M Systematic design of secure mobile grid systems *Journal of Network and Computer Applications* **34**, 1168-1183 (2011)
- [93] Fernandez Munoz & Arteaga Extending a secure software methodology with usability aspects *Position paper in the 3rd Workshop on Software Patterns and Quality (SPAQu 09) in conjunction with OOPSLA 2009* (2009)

- [94] Anon Sturm, P S Validating and implementing security patterns for database applications In *Third International Workshop on Software Patterns and Quality* 40–45 (Orlando Florida, 2009)
- [95] Vivas, J L , Fernandez-Gago, C , Lopez J & Benjumea, A A security framework for a workflow-based grid development platform *Computer Standards Interfaces* **32** 230–245 (2010)
- [96] Georg, G *et al* An aspect-oriented methodology for designing secure applications *Information and Software Technology* **51** 846–864 (2009)
- [97] Mohammad S , Hasheminejad, H & Jalili S Design patterns selection An automatic two-phase method *The Journal of Systems Software* **85** 408–424 (2012) URL <http://dx.doi.org/10.1016/j.jss.2011.08.031>
- [98] Mouatidis, H Giorgini, P & Schumacher, M Security patterns for agent systems *Security* 1–16 (2003) URL <http://hdl.handle.net/10552/384>
- [99] Yoshioka N Washizaki H & Maruyama K A survey on security patterns *Progress in Informatics* **5**, 35–47 (2008)
- [100] Khan, R Secure software development a prescriptive framework *Computer Fraud & Security* **2011** 12 – 20 (2011)
- [101] Fernandez E Yoshioka, N & Washizaki, H Security patterns and quality In *Proceedings of the Third International Workshop on Software Patterns and Quality in conjunction with OOPSLA* 46–47 (2009)

- [102] Kis, M. Information security antipatterns in software requirements engineering. *In proceedings of 9th Conference on Pattern Language of Programs* (2002).
- [103] Schumacher, M. & Fernandez-buglioni, E. *Security Patterns- Integrating Security and Systems Engineering*, vol. 7 (John Wiley and Sons, 2006).
- [104] Ramachandran, J. *Designing Security Architecture Solution* (John Wiley and Sons, 2002).
- [105] Dawson, D. Technical guide - security design patterns. *Journal of Foot and Ankle Surgery* **49**, 566-70 (2000). URL <http://www.ncbi.nlm.nih.gov/pubmed/20801690>.
- [106] Uzunov, A. V., Fernandez, E. B. & Falkner, K. Securing distributed systems using patterns: A survey. *Computers Security* 1-23 (2012).
- [107] Cheng, R. B., Cheng, B. H. C., Konrad, S., Campbell, L. A. & Wassermann, R. Using security patterns to model and analyze security requirements. *In IEEE Workshop on Requirements for High Assurance Systems*, 13-22 (2003).
- [108] Hafiz, M., Adamczyk, P. & Johnson, R. Organizing security patterns. *Software, IEEE* **24**, 52-60 (2007).
- [109] Tryfonas, T. & Kearney, B. Standardising business application security assessments with pattern-driven audit automations. *Computer Standards Interfaces* **30**, 262-270 (2008).

- [110] VanHilst, M., Fernandez, E. B. & Braz, F. Building a concept grid to classify security patterns. In Washizaki, H., Yoshioka, N., Fernandez, E. B. & Jürjens, J. (eds.) *Proceedings of the Third International Workshop on Software Patterns and Quality*, 34-39 (NII, Tokyo, 2009). URL <http://www.grace-center.jp/downloads/GRACE-TR-2009-07.pdf>.
- [111] Yoder, J. & Barcalow, J. Architectural patterns for enabling application security. In *Proceedings of the 4th Conference on Patterns Languages of Programming (PLoP'97)* (1997).
- [112] Kienzle, D. M. & Elder, M. C. Final technical report: Security patterns for web application development (2003).
- [113] Hafiz, M., Johnson, R. E. & Afandi, R. The security architecture of gmail. In *PLoP2004* (2004).
- [114] Schumacher, M. *Security Engineering With Patterns - Origins, Theoretical Model and New Applications* (Springer, 2003).
- [115] Sargon, H. & Carlson. A theoretically-based process for organizing design patterns (2005).
- [116] Mana, A. & Pujol, G. Towards formal specification of abstract security properties. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, 80-87 (2008).
- [117] Schumacher, M. Security patterns and security standards - with selected security patterns for anonymity and privacy. In *Proceedings of the 7th European Conference on Pattern Languages of Programs (EuroPLoP)* (2002).

- [118] Brown L, Diveriti J & Fernandez E The authenticator pattern
Proceedings of Pattern Language of Programs (PLoP) 1999 Conference
(1999)
- [119] Schumacher, M & Utz, R *Security Engineering With Patterns*
(Springer-Verlag New York, Inc , 2003)
- [120] Sandhu, R S Role hierarchies and constraints for lattice-based
access controls In *Proceedings of the 4th European Symposium on Research in Computer Security Computer Security, ES-ORICS '96* 65-79 (Springer-Verlag, London, UK, 1996) URL
http://portal.acm.org/citation.cfm?id=646646_699197
- [121] Sandhu, R S Role-based access control *Advances in Computers* **46**
238-287 (1998)
- [122] Common criteria for information technology evaluation (2006) URL
<http://www.commoncriteriaportal.org/>
- [123] Wille & Gantei *Formal Concept Analysis - mathematical Foundations*
(Springer, Heidelberg 1996)
- [124] Ossher H & Tai P Using multidimensional separation of concerns to
(re)shape evolving software *Communications of the ACM* **44**, 43-50
(2001)
- [125] Parnas, D L Designing software for ease of extension and contraction
In *ICSE 78 Proceedings of the 3rd international conference on Software engineering*, 264-277 (IEEE Press, Piscataway, NJ, USA 1978)

- [126] Sutton, S. M. & Rouvellou, I. Modéling of software concerns in cosmos. In *AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development*, 127–133 (ACM Press, New York, NY, USA, 2002).
- [127] Gagné, D. & Trudel, A. A temporal semantics for workflow control patterns. In *Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, 999–1004 (IEEE Computer Society, Washington, DC, USA, 2008). URL <http://portal.acm.org/citation.cfm?id=1444455.1446218>.
- [128] Stumme, G. Exploration tools in formal concept analysis. *Organic and Symbolic Data Structure: proceedings of the International Conference on Ordinal and Symbolic Data Analysis-OSDA, Paris* 31–44 (1995).
- [129] Stumme, G., Taouil, R., Bastide, Y., Pasquier, N. & Lakhal, L. Computing iceberg concept lattices with titanic. *Data and Knowledge Engineering* **42**, 189–222 (2002). URL <http://portal.acm.org/citation.cfm?id=606457>.
- [130] Boon, C. & Kheng, R. A knowledge-driven model to personalize e-learning. *ACM Journal of Educational Resources in Computing* 1–15 (2006).
- [131] Everts, T. J., Park, S. S. & Kañg, B. H. Using formal concept analysis with an incremental knowledge acquisition system for web document management. In Estivill-Castro, V. & Dobbie, G. (eds.) *Proceedings of*

the 29th Australasian Computer Science Conference vol 48 of *CRPIT*, 247–256 (Australian Computer Society 2006)

- [132] Geid, S , Rafik, T Yevis, B , Nicholas, P & Lotfi, L Fast computation of concept lattice using data mining techniques In *proceedings of 7th Intl Workshop on Knowledge Representation Meets Databases, Berlin CEUR Workshop Proceeding*, 21–22 (2000)
- [133] Liqueie M & Sallantin J Structural machine learning with galois lattice and graphs In *Proc of the 1998 Int Conf on Machine Learning (ICML'98)*, 305–313 (Morgan Kaufmann, 1998)
- [134] Funk, P Lewien, A & Snelting G Algorithms for concept lattice decomposition and their applications Tech Rep Computer Science Report 95-09 (1995)
- [135] Deusen, A V & Kuipers T Identifying objects using cluster and concept analysis In *In 21st International Conference on Software Engineering ICSE-99*, 246–255 (ACM, 1999)
- [136] Snelting G Software reengineering based on concept lattices In *Proceedings of the Conference on Software Maintenance and Reengineering CSMR 00*, 3–10 (IEEE Computer Society Washington DC USA 2000) URL <http://dl.acm.org/citation.cfm?id=518900.795275>
- [137] Schupp, S Krishnamoorthy M , Zalewski M & Kilbide J The “right ’ level of abstraction -- assessing reusable software with formal concept analysis In Angelova G , Corbett D & Pius, U (eds)

Foundations and Applications of Conceptual Structures -- Contributions to ICCS 2002, 74–91 Bulgarian Academy of Sciences (Bulgarian Academy of Sciences, 2002)

- [138] Tonella P & Antoniol G Object oriented design pattern inference In *Proceedings of the IEEE International Conference on Software Maintenance, ICSM '99* 230–238 (IEEE Computer Society, Washington, DC USA, 1999) URL [http //dl acm org/citation cfm id=519621 853393](http://dl.acm.org/citation.cfm?id=519621_853393)

- [139] Godin, R & Mili H Building and maintaining analysis-level class hierarchies using galois lattices In *Proceedings of the eighth annual conference on Object-oriented programming systems languages and applications, OOPSLA '93*, 394–410 (ACM, New York, NY, USA, 1993) URL [http //doi acm org/10 1145/165854 165931](http://doi.acm.org/10.1145/165854.165931)

- [140] Duwell S & Hesse W Bridging the gap between use case analysis and class structure design by formal concept analysis In *Modellierung 2000*, 27–40 (Koblenz, 2000)

- [141] Richards D & Boettger, K Representing requirements in natural language as concept lattices In Bramer, M , Preece, A & Coenen, F (eds) *Research and Development in Intelligent Systems XIA* 425–438 (Springer London 2003)

- [142] Lindig C Concept-based component retrieval In *Working Notes Of The IJCAI-95 Workshop Formal Approaches To The Reuse Of Plans, Proofs And Programs*, 21–25 (1995)

- [143] Mens, K. & Tourwe, T. Mining aspectual views using formal concept analysis. In *Fourth IEEE International Workshop on Source Code Analysis and Manipulation*, 97–106 (IEEE, 2004).
- [144] Francescomarino, C. D. & Tonella, P. Business process concern development and evolution (2008).
- [145] Tilley, T. Formal concept analysis applications to requirements engineering and design. *PhD Thesis (School of Information Technology and Electrical Engineering, University of Queensland)* (2004).
- [146] Snelting, G. & Tip, F. Reengineering class hierarchies using concept analysis. In *Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering, SIGSOFT '98/FSE-6*, 99–110 (ACM, New York, NY, USA, 1998). URL <http://doi.acm.org/10.1145/288195.288273>.
- [147] Snelting, G. & Tip, F. Understanding class hierarchies using concept analysis. *ACM Trans. Program. Lang. Syst.* **22**, 540–582 (2000). URL <http://doi.acm.org/10.1145/353926.353940>.
- [148] Halkidis, S. T., Chatzigeorgiou, A. & Stephanides, G. A qualitative analysis of software security patterns. *Computers Security* **25**, 379–392 (2006).
- [149] Dai, L. & Cooper, K. Modeling and performance analysis for security aspects. *Sci. Comput. Program.* **61**, 58–71 (2006). URL <http://dx.doi.org/10.1016/j.scico.2005.11.006>.

- [150] Rosado, D G , Fernandez-Medina, E Plattm, M & Gutierrez, C Comparison of security patterns *IJCSNS* **6**, 139–146 (2006)
- [151] Mouad A Laveidiere M & Debbabi M An aspect-oriented approach for the systematic security hardening of code *Computers Security* **27**, 101–114 (2008)
- [152] El Yamany H F Capretz M A M & Allison D S Intelligent security and access control framework for service-oriented architecture *Information and Software Technology* **52** 220–236 (2010)
- [153] Mahmoud, Q H Security policy A design pattern for mobile java code In *m Proceedings of the 7th Conference on Pattern Languages of Programming (PLoP 00)* (2000)
- [154] Dong, J , Peng, T & Zhao, Y Automated verification of security pattern compositions *Information and Software Technology* **52**, 274–295 (2010)
- [155] Wolter C Menzel M Schaad A Miseldine, P & Meinel C Model-driven business process security requirement specification *Journal of Systems Architecture* **55**, 211–223 (2009)

List of Publications

1. Sarmah, A.K., Hazarika, S.M., Sinha, S.K.: Security pattern lattice: A formal model to organize security patterns. *In: Proceedings of DEXA 2008* (LNCS 5181, Editors: Sourav S. Bhowmick, Josef Kunj, Ronald R. Wagner), Los Alamitos, CA, USA, IEEE Computer Society (2008) 292–296
2. Sarmah, A.K., Sinha, S.K., Hazarika, S.M.: Exploiting multi-context in a Security Pattern Lattice for facilitating user navigation. *In: Proceedings of 17th International Conference on Advanced Data Communication, Bangalore* (ADCOM, 2009), 215-222
3. Sarmah, A.K., Sinha, S.K., Hazarika, S.M.: An algorithm for navigation in a multi-context concept lattice. *In: proceedings of National Workshop on Design and Analysis of Algorithms, Narosa*(2010), 78-84
4. Sarmah, A.K., Sinha S.K., Hazarika, S.M.: Constraint propagation in workflow - a pattern based approach. *In: Proceedings of National Conference in Machine Intelligence, Narosa* (2011) 134–143
5. Sarmah, A.K., Hazarika S.M., Sinha, S.K.: Composing and maintaining a pattern-based workflow as a constraint graph. *In: proceedings of International Symposium on Cloud and Services Computing, December 17th – 18th 2012, NIT Suratkal* (IEEE Society) (2013) 146-151
6. Sarmah, A.K., Hazarika S.M., Sinha, S.K.: Formal Concept Analysis: Current Trends and Directions. *Artificial Intelligence Review*. (Online First)
7. Sarmah A.K., Sinha, S. K., Hazarika S.M.: A new pattern-based flexible approach for maintaining a constrained workflow. *Accepted for publication in International Journal of Software Engineering and Knowledge Engineering, WorldScientific Press*

Under Review

Sarmah A.K., Hazarika S.M., Sinha, S.K.: Securing A Workflow – When and How ? – *Under review in International Journal of Information and Software Technology, Elsevier Press*