Life Cycle of ROAD Contracts,
Design and Implementation using
Compliance Model

Thesis

Ajay Pal Singh
Supervisor: Alan Colman

Submitted in fulfillment of requirements of
Research Project HIT 4000 for the award of
Bachelor of Computer Science (Honours)

Faculty of Information and Communication
Technologies

Swinburne University of Technology
Abstract

Adaptive software systems development based on software contracts has been a popular approach and is the approach embraced by ROAD framework. Role Oriented Adaptive Design (ROAD) is a meta-model and framework for developing adaptive software systems and is based on dynamic contracts where contracts facilitate implementation of adaptive behavior by fulfilling three functions, i.e. composition, interaction control and measurement of performance over interactions between two participating role players. The design of contracts in ROAD framework is based on a layered model that contains three types of clauses called terms, general clauses and conversation clauses.

In this thesis we design and implement life cycle of ROAD contracts by establishing the effect of violations of contractual requirements on the external representation or state of the contract. We extend contracts defined in ROAD framework and describe relationship between the layers within a ROAD contract. We also introduce compliance model in ROAD contracts in order to facilitate flexible handling of breaches according to the severity of breaches.

This thesis presents formal representation of layers within ROAD contract in XML. The proposed XML representation of terms incorporates the concepts of deontic logic and temporal constraints. We evaluate compliance level of execution of a term by a contracted party by measuring fulfillment of non-functional requirements of terms described in obligations. General clauses further evaluate compliance levels associated with terms and specify the effect of breaches on the contract state. We use ECA (Event-Condition-Action) rules for evaluation and escalation of compliance levels of obligations through layers of ROAD contract. Our compliance model provides ROAD contract designers flexibility in handling contract violations according to the requirements of individual applications. We verify our design by developing a software prototype that demonstrates contract behavior as per our design.
Declarations

This is to certify that this thesis contains no material which has been accepted for the award of any other degree or diploma and that to the best of my knowledge this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Ajay Pal Singh
December 2008
Acknowledgements

I sincerely thank my supervisor Alan Colman for his support and guidance. I deeply appreciate his help and patience throughout the year and for hours of discussions and providing excellent feedback that helped me develop and improve my skills. I will also like to thank Linh Duy Pham for his help at the initial stages of the project and my fellow research student Justin King for help with the setup of test application. I thank all members of CS3 for their guidance and support throughout the year. Finally I will like to thank my family for their unyielding support during the last year that enabled me to complete this project.
# Contents

ABSTRACT...........................................................................................................2

DECLARATION...................................................................................................3

ACKNOWLEDGEMENTS...................................................................................4

CONTENTS........................................................................................................5

LIST OF FIGURES............................................................................................10

1. INTRODUCTION...........................................................................................11

1.2 Research Context - Contracts in ROAD framework.................................11

1.1 Research question........................................................................................12

1.3 Our approach...............................................................................................14

1.4 Thesis outline...............................................................................................15

1.5 Summary.......................................................................................................15

2. LITERATURE REVIEW..................................................................................18

2.1 Motivation for literature review....................................................................18

2.2 Types of Software Contracts.........................................................................19

2.2.1 Interface Contracts...................................................................................19
2.2.2 Electronic contracts for B2B collaboration ........................................... 20
  2.2.2.1 Semantics and Extensions of WS-Agreement ................................. 23
  2.2.2.2 Three layered Architecture for E-Contracts Enforcement ............... 26
  2.2.2.3 Managing Virtual Organizations with Contracts ........................... 28
2.2.3 Contracts in Adaptive Software System Contracts ................................. 31
  2.2.3.1 Developing adaptive systems based on dynamic architectures .......... 31
  2.2.3.2 Software contracts for adaptive software systems ........................ 32

2.2.4 Comparing Adaptive Software Systems contracts, Electronic contracts and
     ROAD Contracts .............................................................................. 34

2.3 Summary ......................................................................................... 35

3. ARCHITECTURE OF ROAD CONTRACTS ................................................. 37

3.1 General Clauses .............................................................................. 38
  3.1.1 Life Cycle of ROAD Contracts ..................................................... 39
    3.1.1.1 Formation ............................................................................. 40
    3.1.1.2 Incipient .............................................................................. 40
    3.1.1.3 Active (Complying) .............................................................. 41
    3.1.1.4 Active (Breached) ............................................................... 41
    3.1.1.5 Suspended ........................................................................... 42
    3.1.1.6 Terminated .......................................................................... 42
    3.1.1.7 Finalized .............................................................................. 43

3.2 Terms .............................................................................................. 43
  3.2.1 Deontic Logic .............................................................................. 44
  3.2.2 Temporal Constraints ................................................................. 45
  3.2.3 Terms – Granting Permissions ..................................................... 46
    3.2.3.1 Expressing constraints on permissions ..................................... 47
3.2.4 Terms – Enforcing Obligations

3.3 Obligations

3.3.1 Types of Obligations

3.3.1.1 Temporal obligations

3.3.1.2 Non-temporal obligations

3.3.2 Aspects of obligations

3.3.2.1 Obligation types

3.3.2.2 Domain variable name

3.3.2.3 Required performance value of domain variable

3.3.2.4 Metric for performance evaluation

3.3.2.5 Comparator

3.3.2.6 Actual performance value

3.3.2.7 Activity Completion variable

3.3.2.8 Status of an obligation

3.3.2.9 Reference to performance provider

3.3.2.9.1 Performance Providers

3.3.2.10 Current compliance level

3.3.2.11 XML representation of obligations

3.4 Evaluating contract compliance

3.4.1 Evaluating compliance in an obligation

3.4.2 Compliance escalation to terms

3.4.3. Compliance escalation to general clauses

3.5 Summary

4. A COMPLIANCE MODEL FOR ROAD CONTRACTS

4.1 Compliance Model

4.1.1 Compliance Function
4.1.2 Compliance Levels

4.2 ROAD contract management with compliance model and ECA rules

4.2.1 ECA Events for ROAD Contracts

4.2.1.1 ECA events for obligations

4.2.1.2 ECA events for terms

4.2.1.3 ECA events for general clauses

4.2.2 Implementing compliance model in obligations using ECA rules

4.2.3 Implementing compliance model in terms using ECA rules

4.2.4 Implementing compliance model in general clauses using ECA rules

4.3 Relationship between compliance levels and contract life cycle

4.4 Self management in ROAD contracts and role played by the ROAD composite organizer

4.5 Summary

5. ROAD CONTRACT TEST APPLICATION

5.1 Contract terms and conditions in plain English

5.2 Representing contract terms and conditions in a ROAD contract using XML

5.3 Implementation of ROAD contract with Java

5.4 Using Drools engine to define rules

5.4.1 ECA rules for general clauses of library-book vendor contract

5.4.1 ECA rules for terms of library-book vendor contract

5.4.1 ECA rules for obligations of library-book vendor contract
5.5 Running the test application..............................................................102

5.6 Summary.........................................................................................108

6. CONCLUSION......................................................................................109

6.1 Contribution...................................................................................109

6.2 Limitations and Future Work.........................................................110

APPENDIX..............................................................................................113

7. REFERENCES.....................................................................................121
List of Figures

FIGURE 2-1: LIFE CYCLE OF EXTENDED WS-AGREEMENT [7]……………….24
FIGURE 2-3: CONTRACT LIFE CYCLE STATE TRANSITIONS IN [17] .................29
FIGURE 3-1: UML DIAGRAM OF ROAD CONTRACTS.................................38
FIGURE 3-2: LAYERS IN ROAD CONTRACTS.............................................40
FIGURE 3-3: LIFE CYCLE OF ROAD CONTRACT........................................43
FIGURE 3-4: XML REPRESENTATION OF A TERM WITHOUT ANY ASSOCIATED OBLIGATIONS.................................................................47
FIGURE 3-5: XML REPRESENTATION OF TERM WITH ASSOCIATED OBLIGATIONS.................................................................................................49
FIGURE 3-6: STATUS OF AN OBLIGATION..................................................56
FIGURE 3-7: ROLE PLAYED BY PERFORMANCE PROVIDERS IN COMPLIANCE EVALUATION.................................................................59
FIGURE 3-8: REPRESENTATION OF OBLIGATION IN XML.........................60
FIGURE 3-9: OBLIGATION XML SCHEMA................................................61
FIGURE 3-10: ESCALATION OF COMPLIANCE WITHIN THE ROAD CONTRACTS......................................................................................................65
FIGURE 4-1: COMPLIANCE EVALUATION WITH A COMPLIANCE FUNCTION........................................................................................................70
FIGURE 4-2: ECA EVENTS FOR ROAD CONTRACTS......................................77
FIGURE 4-3: COMPLIANCE EVALUATION IN OBLIGATION USING ECA RULES..................................................................................................79
FIGURE 4-4: COMPLIANCE ESCALATION BY TERMS USING ECA RULES.....80
FIGURE 4-5: EVALUATING EFFECT OF COMPLIANCE LEVELS BY GENERAL CLAUSES USING ECA RULES..................................................81
1. Introduction

Current software systems lack ability to deal with environmental variations as computing environments become more open, dynamic and distributed. Also the goal of current systems is fixed at design time rendering them incapable in adapting to changing user requirements at runtime. These limitations in the current software systems give rise to the need for adaptive software that can survive in dynamic environments and adapt to changing requirements. An adaptive system should be flexible enough to deal with variations in heterogeneous environments and still achieve system level goals where these goals are subject to changes as well. Some of the current frameworks proposed for the creation of adaptive software have been based on software contracts where contracts are used to manage dynamic software architectures. In this study we define and implement the life cycle of contracts used by ROAD framework for developing adaptive software systems. The research aims at defining the life cycle states of the ROAD contract, execution and monitoring of transition of states and management of life cycle by the contract itself and ROAD composite organizer.

1.1 Research Context - Contracts in ROAD Framework

Role Oriented Adaptive Design (ROAD) proposes a meta-model and a framework for creation of an adaptive software system. The adaptive software applications are defined by ROAD framework as networks of functional roles which are executed by players (objects, components, services, agents, people, or role composites). These flexible organizational structures are adaptive because the relationships (contracts) between roles, and the bindings between roles and players, can be regulated and reconfigured at runtime. The basic underlying principles in ROAD Framework are separation of control from the process, distribution of control down through a recursive structure and separation of definition of role from its implementation.

ROAD is based on dynamic contracts where contracts not only compose and control the interactions between role-players but also measure the performance of those interactions.
ROAD Contracts are defined by Colman in [1] as binary association classes that express the obligations of the contracted parties to each other. Further they are not only a specification of mutual obligations but also monitor runtime execution of contract terms to establish compliance with the specification. Dynamic contracts are central to ROAD framework.

ROAD framework enables development of software that is able to achieve varying system level goals in dynamic environments. ROAD Framework achieves adaptive behavior by monitoring and manipulating contracts, along with the role-player bindings. Contracts and the Organizer of the ROAD Composite regulate relationships between role players so that the role players are held accountable for their performance. Contracts prevent unauthorized or invalid interactions and also take appropriate action in case of an under performance by a role player. Contracts in ROAD Framework fulfill three functions i.e. composition, interaction control and measurement of performance over interactions between two participating role players. ROAD contracts further contain three types of clauses called terms, general clauses and conversation clauses. Terms of a contract specify allowed atomic interactions between contracted parties. Further terms have obligations associated with them that specify non-functional requirements of each functional interaction. General Clauses in a contract define life cycle of a contract, hence defining conditions relating to initiation, continuation and termination of a contract. Conversation clauses specify the sequence to be followed by the contracted parties while executing contract terms.

1.2 Research Question

In ROAD Framework, contracts should not only specify the functional interactions between contracted parties but also act as a runtime entity that monitors and to some degree enforces those interactions. In order to realize such rich functionality within a contract, functional and non-functional interactions between the parties must be clearly defined, the rules that define the pattern of those interactions expressed, the effect of those interactions on the contract state and any other actions resulting from those effects described. ROAD Framework takes an architectural approach to contract design that
facilitates runtime monitoring and enforcement in addition to identifying the components of a contract by logically dividing ROAD Contracts into three layers consisting of general clauses, conversation clauses and terms. But it provides only the basic architecture to realize such design. The framework does not elaborate on the relationship between these layers to describe how changes in one layer affect other layers. A ROAD contract like manual contracts goes through various states as it is created, negotiated, enacted and finalized. ROAD contract needs explicit representation of contract states as each runtime state in the life cycle of contract represents state of fulfillment of the contract at any particular point or we can say that each state in the life cycle of a ROAD contract is an external representation of its current internal state. The existing definition of Contract life cycle is preliminary and triggers for state transitions have not been stated. Even though the current design states that the breaches in terms of the contract will change contract state to some kind of violated state but how this will actually occur is not described. Further the management of life cycle of ROAD contracts between contract itself and composite organizer is not explained. In order to tackle the above problems, this thesis addresses the following questions.

- How do general clauses, terms and obligations in a ROAD contract relate to one another, how to recognize that the change has occurred in one layer and how to escalate the change to the other layers?
- When contracted parties fail to fulfill their contractual requirements, how does this affect the contract as a whole, i.e. affect of the breaches on the external representation or state of the contract?
- How can a ROAD contract flexibly handle contract breaches according to precise business requirements?
- How is the contract life cycle managed between the general clauses inside contract and ROAD composite organizer?
1.3 Our Approach

This section outlines our approach for solving the above mentioned problems. ROAD Framework adopts a generalized layered model for ROAD contracts. Our approach extends the current specification as we present a formal representation of layers within a ROAD contract, i.e. General Clauses, Terms and obligations using XML. The proposed XML based formal representation of terms and obligations incorporates the concepts of deontic logic and temporal constraints. General clauses are represented as a set of ECA (Event-Condition-Action) rules (McCarthy and Dayal [11]).

We also use ECA rules to establish relationship between the layers of ROAD contract. We illustrate how the current state of ROAD contracts is evaluated by the general clauses from the current performance levels of its terms. Proposed system uses ECA rules for triggering evaluation of performance at obligations and then escalation of evaluated performance through the layers of the contract, i.e. from obligations to terms and to general clauses. General clauses are defined as a set of ECA rules that are triggered with events. On being triggered general clauses evaluate the reported performance results and take further action or set of actions depending on the outcome. One of the actions could be changing the external state of the contract to reflect current internal conditions of the contract.

In order to enable flexible handling of contract breaches we introduce compliance model in ROAD contracts. The compliance model generates compliance level when actual performance is measured against required performance. This compliance level can be configured by contract designer or composite organizer according to the precise business requirements. We then illustrate how general clauses use compliance levels to work out the external state of a contract. We also describe how general clauses and ROAD composite organizer collaborate to control the life cycle of ROAD contract.

In order to verify our design we provide a java based test application that demonstrates the implementation of our design. We use JAXB to generate java classes from XML schema of ROAD contract. We populate java objects with data at runtime by
unmarshalling ROAD contract XML file using JAXB API. In order to implement ECA rules in ROAD contracts we use Drools rule engine as it provides support for declarative programming. In our application a chain of rules is triggered for evaluation and escalation of compliance results from obligations to general clauses. General clauses are also implemented as a set of rules that are triggered in response to reported compliance results. Different rules are triggered in response to different compliance levels and some of these rules transform the external state of the contract. The test application generates different compliance levels when different actual performance values are passed to the contract at runtime. The generated compliance levels correspond to different performance threshold values specified in the XML representation of obligations.

1.4 Thesis outline

This thesis is arranged as follows:

Chapter 2 provides background to this study by starting with the introduction to the types of contracts used in the field of software engineering. We describe our motivation for the literature review and explain the basis on which we chose the literature to be reviewed. We then describe electronic contracts used in B2B environments and provide discussion about some of the characteristics of electronic contracts in the current literature. We also present a detailed analysis of contract life cycle management in three electronic contract specifications. This is followed by discussion of adaptive software systems and adaptive software systems with dynamic architectures. This discussion provides an insight into the role of contracts inside adaptive software systems where contracts are used to manage relationships between components in dynamic software architectures. We then examine the role of contracts in three contract based adaptive software system frameworks. The following section presents comparison of electronic contracts, adaptive software system contracts and ROAD contracts. In the end we present a comparative table that summarizes the features of reviewed electronic contracts.

Chapter 3 starts with the introduction to the architecture of ROAD contracts. We then present our approach and elaborate on the roles played by each layer of the contracts. We
present formal representation of terms and obligations inside ROAD contracts that is based on the concepts of deontic logic and temporal constraints. We describe the types of obligations that can be associated with terms and elaborate in great detail on how obligations express and evaluate compliance for performance requirements of terms. We conclude this chapter by illustrating how the contract performance is evaluated by using obligations, terms and general clauses.

Chapter 4 introduces our proposed compliance model for ROAD contracts. We illustrate on how different compliance levels can be set inside compliance functions attached to the obligations of terms and how compliance functions provide flexibility in dealing with violations to contract terms. We then introduce ECA (Event-Condition-Action) rules and describe how ECA rules facilitate escalation of compliance level through the contract hierarchy. We also describe how compliance model provides variable life cycle to ROAD contracts. We conclude this chapter by describing how general clauses and ROAD composite organizer together manage the life cycle of ROAD contracts.

Chapter 5 provides details of our test application that implements compliance model in ROAD contracts. The test application verifies the solution proposed in this thesis.

Chapter 6 concludes the thesis. This chapter provides discussion about contribution of this thesis to the ROAD framework. We also discuss limitations of this study and required future work.

1.5 Summary

This chapter outlined the research context of this thesis. We described the problems addressed by this thesis and our approach for tackling those problems. In the end we presented the description of the structure of the thesis that also outlined the contents of each chapter.
2. Literature Review

In this chapter we provide background to our work. We start with the discussion of types of software contracts used in the field of software engineering. We provide a brief overview of interface contracts followed by detailed discussion of electronic contracts that represent real world business relationships as software objects and contracts used in adaptive software frameworks that also includes ROAD contracts. We examine some of the current research efforts that focus on the later two categories. In the discussion we focus on the concepts used in these approaches for the design and implementation of contracts and particularly life cycle of contracts. We also examine how contracts in these frameworks respond to events in different phases of their life cycles. We then compare these two types of contracts with each other and with ROAD contracts. We conclude this chapter with a comparison that summarizes the various characteristics of reviewed contract frameworks.

2.1 Motivation for this Literature Review

We identified three main categories of software contracts in the field of software engineering as: Interface Contracts, Electronic Contracts for B2B collaboration and Software Contracts for adaptive software architectures. In this chapter we limit our discussion to the later two as Interface contracts are completely different from ROAD contracts as these types of contracts take into account the requirements of only one contracted party. ROAD contracts on the other hand encapsulate obligations and requirements of both contracted parties. The field of electronic contracts as such has also attracted a lot of research attention in the past few years with emphasis into contract creation, negotiation, enactment and enforcement aspects. This literature review only examines those research efforts that focus on monitoring and enforcement aspects or provide some insight into the life cycle management of electronic contracts. We also discuss contract based adaptive software architectures that describe contract behavior in response to changes in requirements and operational environments and present some details about monitoring and enforcement aspects of contracts. We only briefly examine
proposed frameworks under this category as these frameworks are more focused on achieving runtime adaptation for the proposed software systems and do not elaborate enough on the design and implementation of their contracts.

2.2 Types of Software Contracts

Software engineering is very familiar with the concept of software contracts though these contracts have been used in a variety of ways. Interface contracts have been used to specify rules of interaction from a single software component’s view that must be followed by the other component while other component is anonymous. As ROAD contracts are two party contracts that express obligations of both contracted parties, this class of software contracts holds little relevance to our research. Another use of software contracts is in dynamic architecture based adaptive software systems where contracts exercise control on component relationships and facilitate re-configuration operations in order to adapt the software system to changing requirements or environments. The third category of software contracts is electronic contracts or sometimes called E-Contracts. Goal of electronic contracts is to automate various phases of manual contracting processes providing support for automated negotiation, enactment, monitoring and enforcement. In the next section we briefly discuss interface contracts and provide detailed discussion of other two categories in the following sections.

2.2.1 Interface Contracts

Interface contracts have been widely used to codify the rules of using a software component in order to facilitate interaction between software components. This category of software contracts specifies pre-conditions, post-conditions and invariants that must hold to enable a given interaction with a software component. Therefore this class of software contracts is used to express interaction constraints among software components from a single component’s view. ROAD contracts are completely different from interface contracts as ROAD contracts are dynamic entities that not only express interactions constraints for both contracted parties but also measure the performance of those interactions. We therefore do no discuss interface contracts any further and focus on the
other two categories that have more similarities with ROAD contracts as discussed in the following sections.

2.2.2 Electronic contracts for B2B collaboration

The emergence of e-commerce has successfully changed the conventional business models providing businesses with opportunities to reach more customers with minimal costs and have provided new models for B2B and B2C collaborations. Legal contracts have been used since ancient times in the form of hard copy contracts for establishing various business collaborations and partnerships. With the emergence of internet based commerce, electronic contracts have been proposed to enable businesses to electronically manage distinctive business relationships with different business partners in order to allow automated collaboration between businesses. Electronic contracts are defined by Molina-Jimenez [12] as electronic form of conventional contracts consisting of one or more executable files and optionally containing some more text, graphic or images files enacted together to enforce what English contracts lay down. The objective is to move paper contracts to an electronic format and use third party systems or sub-systems built within organizations to facilitate various contractual activities. Electronic contracts automate various aspects of the traditional paper contracts and regulate business interactions to ensure legality and protection of all parties involved in B2B e-commerce.

Much literature is available [3, 5, 7, 8, 9, 13 and 22] that provides specifications for enactment, enforcement and run-time monitoring of electronic contracts. Most of this literature relies on using third party middleware services for runtime support of electronic contracts. Some of these services include contract monitors, negotiators and other services provided by e-market places like service registries. Requirement for standard contract templates consisting of typical clauses that are applicable to most contracts and that can be extended to capture any domain specific requirements if necessary is frequently advocated in the literature.

Zoran Milosevic in collaboration with other authors presents his work in this field in [5, 8, 13, 23 and 24]. Marjanovic and Milosevic [5] present a formal model for
representation of electronic contracts. The paper proposes that the representation of terms of a contract need to encapsulate the concepts of temporal and deontic logic. The paper also provides formal expressions for the terms of contracts that capture the above principles. But the scope of this paper is limited to modeling of electronic contracts only and paper didn’t look into the aspects of monitoring, enforcement or life cycle of contracts. Goodchild et. al. [8] outlines approach for specification, implementation, monitoring and enforcement of B2B electronic contracts by identifying various roles such as Contract Repository, Contract Monitor, Contract Enforcer, Contract Validator and Notary. Contract is eventually represented as an XML document that can be enforced by using the services provided by above roles. The roles described above are played by independently deployed components unlike ROAD contracts where contract encapsulates monitoring and to some extent enforcement. This paper also did not look at life cycle of the electronic contracts at all. Governatori et. al. in [23] presents logic based formalism to establish compliance of business processes with business contracts. Governatori and Milosevic in [24] provide contrary-to-duty structures as alternative obligations when original contractual obligations are violated by a contracted party. Milosevic et. al. [13] presents an approach towards automated contract enforcement based on five levels of compliance. The different compliance levels in [13] signify the ability of the system to apply corrective measures after a breach has occurred. Our approach complements the work in [13] as we specify how to capture different levels of breaches in a generalized model of compliance levels while authors in [13] provide a framework to handle the breaches by providing a range of enforcement mechanisms.

Frankove et. al. [7] working in the domain of Service Oriented Computing extend WS-Agreements from mere textual descriptions to agreement capable of being monitored at runtime to establish compliance. The paper extends WS-Agreements to enable handling runtime breaches in agreement terms in a more graceful manner than terminating the agreement at once after a breach occurs and to achieve that they introduce the notion of negotiation terms that enables modification of QoS at runtime for handling recoverable violations to contract terms. This paper in general pays adequate attention to the life cycle of the extended WS-Agreement model but did not elaborate enough on how to actually
monitor runtime performance of contracted parties as it relies on some third party monitoring system.

A multi-layered architecture for enforcing E-Contracts between two or more contracted parties is presented by Chiu et. al. in [3]. E-contracts are defined in [3] as a computerized facilitation or automation in a cross-organizational business process. This study briefly describes contract life cycle and presents contract clauses based on Event-Condition-Action rules where events are monitored and appropriate action is taken by the affected party if required.

Metso and Kutvonen [9] propose multi-party contracts that not only record the contract terms and non-functional requirements but also encapsulate role descriptions for fulfilling contracts. The framework presented is based on B2B middleware services provided by web-Pilarcos architecture [10]. In this approach compliance is established by verifying messages passed between the parties against the required message types under the current state of the business process. This paper did not classify breaches into different levels but proposed configuring the monitor object (Responsible for monitoring contract execution) to proactive, active or passive behavior in order to provide flexibility in escalation of breaches.

Daskalopulu and Maibaum [6] specify contracts based on modal action logic and deontic logic. Contract terms are presented using deontic and temporal constraints and cascades of contrary-to-duty structures are proposed based on above presentation as error recovery mechanisms after capturing contract breaches according to the severity of breach. Again though paper talks about contract states and transaction states but complete contract life cycle is not defined and related to contract breaches.

Therefore though much work [3, 5, 8, 9, 13 and 22] provides details about representation of electronic contracts to facilitate monitoring and enforcement but these approaches do not pay any or enough attention to relationship between contract violations and its life cycle states. Though Frankova et. al. [7] describes contract life cycle states based on the
current status of functional and non functional requirements, the paper did not show how the status of these requirements is evaluated. These approaches typically rely on external components or built in sub-systems to enable runtime monitoring and enforcement and do not describe how that actually occurs. ROAD contracts on the other hand monitor and to some extent take enforcement actions themselves after evaluating predefined conditions. In our research we not only establish relationship between breaches in contract terms and contract life cycle we also demonstrate how breaches are captured. We continue this discussion by looking at the three specifications [3, 7 and 9] in detail. We chose these specifications for detailed discussion as these approaches provide best insight into various aspects of complete life cycle management of electronic contracts and to some degree also describe how contracts in these frameworks respond to various events at runtime.

2.2.2.1 Semantics and Extensions of WS-Agreement
Frankova et. al. in [7] proposes extensions to WS-Agreement specification to include runtime re-negotiations of agreement terms and more efficient handling of agreement term violations to capture minor breaches in a more moderate fashion. The author highlights the weaknesses of the current specification that terminates the agreement immediately after violation of a single term of the agreement and suggests that more graceful degradation is required instead. Author also argues in the paper that agreements according to current specification are only a vague textual overview of the intended meaning of the agreement. The paper provides a formal definition of agreements and a specification for analyzing evolution of agreements through its life cycle in order to overcome this shortcoming. This study defines different life cycle models for agreements, terms of agreements and non-functional requirements of the terms of the WS-Agreement. The author extends the current specification of WS-Agreements to include the processes of runtime negotiation and a warning facility to handle minor breaches in terms of the agreement. The extended life cycle of agreements is presented in the figure 2-1 below.
Paper defines life cycle states of agreements as external states as these are observed externally. The contract is in negotiated state during the negotiation process and enters not_observed state when negotiation is finalized and accepted by the contracted parties. The contract is ended if parties fail to negotiate a desired agreement. The paper differentiates not_observed from observed state as a service is only monitored or observed after services start executing and not immediately after agreement implementation. As soon as a service execution begins, agreement enters observed state and progresses to checked state as monitoring system checks provided services against QoS requirements. To implement proposed extensions of handling violations more gracefully, the author introduces new agreement terms called negotiation terms. The negotiation terms specify new conditions that enable run-time modification of QoS. Any violations detected at this state result in progression to warned state if the violation is anticipated but hasn’t actually occurred yet and re-negotiation state if the violation is detected but can be handled by re-negotiations with the defaulted party. If a negotiation term exists in relation to the violated QoS, only then agreement state proceeds to re-negotiation state. If a negotiation term is not found for a violated term, then agreement state is changed to denied and then to finished. The extension also provides a facility for changing QoS through renegotiations if desired. But all renegotiations can only be carried
out if a negotiation term that relates to violated QoS has already been agreed upon by the parties during initial negotiation. The definition of internal life cycle of agreement is given by combining life cycles of terms of the agreement and QoS related to those terms. The internal life cycle states of an agreement proposed in the paper are given below in table 2-1. In the first state, not_ready implies to term not being ready and not_determined for QoS that has not been set by the parties yet. The definition of the other internal states follows in the same way.

<table>
<thead>
<tr>
<th>Term State</th>
<th>QoS State</th>
<th>Term State</th>
<th>QoS State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Not ready</td>
<td>7</td>
<td>Stopped</td>
</tr>
<tr>
<td></td>
<td>not_determined</td>
<td></td>
<td>negotiated</td>
</tr>
<tr>
<td>2</td>
<td>Ready</td>
<td>8</td>
<td>Stopped</td>
</tr>
<tr>
<td></td>
<td>not_determined</td>
<td></td>
<td>fulfilled</td>
</tr>
<tr>
<td>3</td>
<td>Ready</td>
<td>9</td>
<td>Stopped</td>
</tr>
<tr>
<td></td>
<td>negotiated</td>
<td></td>
<td>violated</td>
</tr>
<tr>
<td>4</td>
<td>Running</td>
<td>10</td>
<td>Stopped</td>
</tr>
<tr>
<td></td>
<td>fulfilled</td>
<td></td>
<td>non-recoverably_violated</td>
</tr>
<tr>
<td>5</td>
<td>Running</td>
<td>11</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td>violated</td>
<td></td>
<td>fulfilled</td>
</tr>
<tr>
<td>6</td>
<td>Running</td>
<td>12</td>
<td>Finished</td>
</tr>
<tr>
<td></td>
<td>warned</td>
<td></td>
<td>violated</td>
</tr>
</tbody>
</table>

Table 2-1: Twelve possible combination of states of terms and associated QoS requirements

On one hand this paper in its extension to WS-Agreement provides a great detail on evolution of WS-Agreement through its life cycle states but on the other hand details about capturing violations that trigger state transformations are missing. The reason behind author’s decision to have an observed state and then another checked state in the life cycle of agreements is also not clear as agreements should be monitored for compliance all the time while the services are running. This work relies on a third party monitoring service for monitoring WS-Agreements so author did not provide any method for monitoring service executions for compliance in this paper. ROAD Contracts do this monitoring by matching the current performance of contracted parties to the agreed performance stored inside obligations. This paper also proposes runtime renegotiation of agreements by matching violated QoS to pre-agreed negotiation terms but again details about how this match will be performed are missing.
2.2.2.2 Three-Layer Architecture for E-Contract Enforcement in an E-Service Environment

Chiu et. al. [3] proposes multi-layered architecture for enforcing E-Contracts between two or more contracted parties. The paper argues that cooperation and trust between contracted parties where a party can monitor the performance of counter parties is essential for future success of E-Services. The architecture briefly describes Contract life cycle with four states: Information Exchange, Negotiation, Enactment and Enforcement. During the information exchange phase service providers advertise their services while service consumers look for the appropriate provider. During negotiation phase the agreed values for contract clauses are set and agreed upon by all contracted parties. The enactment phase goes from initiation of contract to execution of business activities by the contracted parties.

For enforcement of E-Contracts the paper proposes a three-layer architecture comprising of Document Layer, Business Layer and Implementation Layer. Document layer specifies contracts in terms of three types of clauses: obligation, permission and prohibition. To enforce contracts, the variables in contract clauses need to be monitored as their values change due to execution of business activities. The paper argues that constant monitoring of these variables to recognize any breach is not viable as it incurs massive overhead on the system. Instead the paper proposes transformation of contract clauses in document layer to a set of ECA (Event, Condition, Action) rules in the business layer. Using ECA rules only events are monitored as they occur and appropriate action is taken by the affected party depending on the current conditions. These conditions are logical expressions and can be evaluated according to provided algorithms. Enforcement action will be executed only when condition is satisfied that may lead to other events and actions otherwise an exception will be raised. The paper argues the need for temporally bound actions when actions are an obligation of a contracted party otherwise enforcement of such obligations is not possible. Further the contracted parties are responsible for raising events as they occur and pass the event information to another party if the other party has subscribed for the event. Some of these events can be temporal that may also trigger certain actions. Contract Clauses in Document layer are mapped to ECA rules.
using set of logical expressions in deontic logic. The implementation layer of the contracts comprises of action implementation and cross organizational communication. The paper outlines implementation of action with EJBs and cross organizational event interfaces with Web Services.

Figure 2-2: E-Contract Meta-Model of Contract Enforcement in UML [3]

The E-Contracts defined in this paper are very different to ROAD contracts though both are trying to accomplish similar goals. ROAD Contracts are dynamic entities where its clauses can be changed to suit current requirements without any restrictions whereas E-Contracts have certain fixed template variables limiting its adaptability. Further ROAD contracts unlike E-Contracts are not multi-layered in the same sense (ROAD contracts have layers of components within i.e. general clauses, conversation clauses and terms) and can be bound to only two parties at one time. Also the contract and composite organizer in ROAD framework are responsible for monitoring performance of contracted parties removing the need for contracted parties to monitor events in counter parties as in
E-Contracts. E-Contracts in this paper depend on the services of contract enactor subsystem for performing regular business functions and contract enforcer subsystem to detect contract breaches on behalf of contracted parties. The paper did not pay much attention to the contract life cycle though it proposes ECA rules that allow contracts to respond to breaches as they occur. Therefore though the paper provides details about monitoring of contract breaches with ECA rules but did not attempt to examine the effect of breaches on the life cycle of the contracts.

The paper states that formulation of generalized measures to handle contract breaches is not possible as domain specific knowledge is necessary to handle any particular breach. We suggest that handling of breaches to the point of effect of breaches on the life cycle of contracts can be generalized as we illustrate later with application of compliance model to ROAD contracts. Any recovery actions or applications of penalties on the party responsible for breach cannot be generalized as these actions depend on the requirements of individual applications.

### 2.2.2.3 Managing Virtual Organizations with Contracts

Metso and Kutvonen [9] provide a framework for managing virtual organizations with electronic contracts. The framework presented is based on B2B middleware services provided by web-Pilarcos architecture (Kutvonen et. al. [10]). Web-Pilarcos architecture provides middleware facilities for managing contracts, inter-enterprise communication and monitoring contracts at runtime through a Contract Object, NetworkManagementAgent (hereafter referred to as Agent), and Monitor object respectively. Some other service components provided by this architecture to support the framework at runtime are a Populator (for lookup and interoperability checking of potential services), TypeRepository, ServiceOfferRepository and BusinessNetworkModel. The role of an Agent is to provide a central interface to an organization, configure the local middleware platforms and communicate between the contracts and internal applications during contract life cycle transitions. Contract object is an active distributed object that controls its own life cycle and encapsulates all the contract agreement information including information about the contracted parties and
organizational signatures to verify the legitimacy of the sent messages during contract execution. Monitors provide contract enforcement service. Monitors are plugged into the communication channel and regulate the service behaviors of contracted parties according to the recorded contract terms. The monitoring service intercepts every message sent between the parties and establishes compliance with the pre-defined allowed message types for a current business process state. Further monitors can be configured to be pro-active, active or passive. Pro-active monitor blocks every non-compliant message, active informs the Agent while Passive only records the breach. Other than reporting breaches, monitor object also reports progress of the current task by reporting current state of the task being executed. Valid task states are started, not started, finished, and failed. The figure 2-3 below presents the life cycle of contract specified in this approach.

![Figure 2-3: Contract Life Cycle State Transitions in [17]](image-url)
The contract life cycle begins as contract is populated by the initiator or organization creating the virtual organization. After entering the required services or term definitions, the contract draft is sent to potential service providers and the contract enters negotiation state. Parties can accept or counter propose or reject the proposal. Contract moves to negotiated state as the negotiation is finalized. Negotiated state captures the time in the life cycle of contract when the contracted parties are getting ready for fulfilling their roles by configuring themselves as per the requirements of the contract. Established state is reached as soon as the contract execution begins. Contract is monitored by the monitoring service to determine compliance by contracted parties according to contract terms and if non-compliance is detected the contract state is changed to unusable and further actions initialized to revalidate the contract. The contract moves back to the established state if non-compliance issue is resolved with the defaulted party. If a participant withdraws from the contract possible due to its inability to provide the service required, the contract is moved to reorganization state. During reorganization state, the contract is re-populated and new party is searched that can fill the vacant role. If a new party cannot be found the contract is terminated. As new parties are attached to the virtual organization or a party is detached, this changes the structure of the organization.

This approach proposes multi-party contracts that not only record the contract terms and non-functional requirements but also encapsulate role descriptions for fulfilling contracts. Though the author proposes that runtime compliance is monitored by a monitor object that monitors runtime behavior of contracted parties but the paper did not elaborate on the details about how monitor object actually works. ROAD Contracts on the other hand are two party contracts and monitor run-time compliance to the contract terms themselves. Also ROAD Contracts only contain references to the role descriptions of the potential or bounded role players. In general this approach provides a good overall architecture for creation of virtual organizations connected by contracts. But the specification of contract life cycle states and state transitions did not define contract states corresponding to contract functioning under a low level breach. Though monitor object can be configured to be pro-active, active or passive but these configurations only specify how monitor object treats every breach. This approach doesn’t allow classification of breaches
according to their severity. Breaches in the terms of contract will not have same kind of implications. Very serious breaches may stop the contract execution instantly while others may only be recorded. Frankova et. al. in [7] proposed negotiation states as an extension to WS-Agreements for ability to deal with low level breaches in a moderate fashion but negotiation terms for those breaches had to be pre-defined.

### 2.2.3 Contracts in Adaptive software systems

The two most important challenges for implementing adaptive software in dynamic and distributed environments are altering service provision according to varying user requirements and handling unreliable and dynamic environments. Adaptive software system needs to maintain service provision to the user while existing components become unavailable or their performance degrades by finding new components. In other words, adaptive software systems need to be able to alter their structure in order to sustain required performance levels. Colman [1] presents ROAD Framework as an Adaptive Software System capable of adapting to variations in execution environments and changes in user requirements by altering its structure.

#### 2.2.3.1 Developing adaptive systems based on dynamic architectures

Garlan and Perry define software architecture in [20] as “the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time”. Dynamic architectures can be defined as architectures where the software structure can be dynamically altered by recomposing/revoking existing relationships between existing components or with new components that better satisfy the requirements of the system. Dynamic software architecture as a basis for development of adaptive software systems has been a popular approach adopted by many research efforts [1, 14, 15, 16, 18 and 19]. Colman categorizes dynamic architecture based approaches to building adaptive software into three groups terming them as structure oriented frameworks, control oriented frameworks and contract oriented frameworks and adopts the last approach for ROAD framework. The contract based adaptive architecture proposed in [1] provides adaptability to ROAD composite by dynamically changing the structure of the composite by having different
services playing roles at different times and providing ability to not only create new role instances but to revoke/create contracts between existing roles.

**2.2.3.2 Software Contracts for adaptive software systems**

A few approaches to building adaptive software systems have been based on contracts where contracts are used to administer control over structure by dynamic specification of relationship between system components. Research efforts in [1, 14, 15 and 16] propose contract based architectures for adaptive software systems. All of these approaches are component based and use contracts to control interactions and relationships between the software components. The components in these systems are loosely-coupled and contracts are used to regulate their mutual relationships and behaviors. We provide an overview of the three contract based adaptive software systems below:

Chang et. al. [15] presents a framework called ConFract, a contract-based framework for hierarchical components in which contracts are runtime objects that are dynamically built from specifications, and automatically updated according to dynamic reconfigurations. This framework is based on the fractal component model presented by Bruneton et. al in [21]. The paper describes rich contracts between software components that not only fulfill functional requirements at assembly time and runtime but also take care of QoS requirements. ConFract defines three types of contracts. Interface Contracts responsible for binding between the interfaces of a client and a server. External composition contracts define usage and external behavior of the composite with respect to its environment. Internal composition contracts express how a composite is assembled and define internal behavior rules of the implementation of the composite component. These contracts are managed by Contract Controllers that negotiate alternative ways of satisfying the requirements when breaches occur. When a breach occurs in the provision of contract terms, contract controllers initiate a negotiation process with other components. The other components propose modifications by reconfiguring themselves that are then analyzed by the contract controller to establish a validity check. If a proposal is found to be valid, negotiation is finalized and valid state is restored. In ConFract contracts are executed and monitored by the contract controllers unlike ROAD Contracts that manage themselves (to
some extent). This study describes the negotiation phase in the life cycle of contract controller after breaches occur in provision of services but did not elaborate enough on the details of other states in the life cycle. The details about monitoring of breaches are also missing.

Contract based Adaptive Software Architecture (CASA) proposes a framework where software systems can adapt to the changes in execution environments due to variations in available resources. In this architecture presented by Mukhiza et. al in [14], the application changes behavior in response to the changes in resources available by re-configuring itself at runtime and uses application contracts to specify its adaptation policy. CASA Contracts that operate in different zones manage adaptation at runtime by mapping the service requirements (From Service Negotiator) to the available resources. If the resources are available, contract looks for the specified configuration of system components for those resources. Each contract zone differs in level of service provided. Each zone further contains a list of component configurations available in that zone and their corresponding resource requirements. This framework did not attempt to explicitly define the life cycle of its contracts. The operating zones of CASA application contracts distinguished by the service level implicitly capture a particular life cycle state. Each zone varies in the level of service provided or expected by the corresponding application. This approach only gives preliminary details about the design of contracts used in the framework.

To realize dynamic software architecture, Dowling and Cahill in [16] propose K-Component architecture meta-model. This specification uses adaptation contracts to encapsulate architectural constraints of the meta-model. These contracts are meta-level objects and can be loaded or unloaded at execution time. Further adaptation contracts are reflective programs as they are responsible for invoking reconfiguration operations on the architecture meta-model resulting in modifications of the software architecture itself. This framework similar to ROAD Framework separates functional system from the management system. This system achieves adaptation through adaptive contracts that take two forms: Configuration Contracts and Connector Contracts. Configuration
Contracts are responsible for mapping adaptation requirement to the associated reconfiguration operations. These contracts are made of series of conditional statements that test for occurrence of adaptation events and trigger associated reconfiguration operations. Connection Contracts define the architectural constraints for external associations of system components with external systems. The study did not attempt to explicitly or implicitly define life cycle of any of its contracts.

It follows from above discussion that adaptive software system approaches focus on achieving adaptation for the software system in response to changing environments and sometimes requirements. These approaches give very marginal description of contracts used in their frameworks as they do not elaborate much on design and implementation of their contracts. The life cycle of contracts is either not discussed at all or enough details are missing. These frameworks also did not attempt to establish the relationship between contract breaches and life cycle of contracts. We included this discussion here as ROAD contracts perform similar functions in ROAD framework. ROAD contracts also facilitate implementation of adaptive behavior in ROAD framework. ROAD Framework achieves adaptive behavior by monitoring and manipulating these contracts, along with the role-player bindings. In the next section we present a comparison between electronic contracts for B2B collaboration and adaptive software system contracts.

2.2.4 Comparing adaptive software system contracts, electronic contracts and ROAD contracts

In this chapter we examined literature on electronic contracts and adaptive software system contracts. Both contract types govern structures of electronic business models for inter-organizational interactions.

Electronic contracts enable automated business collaborations. Some of the discussed electronic contracts provide a way of specifying contract terms between the contracted parties by giving them formal definitions that capture the deontic and temporal constraints that govern interactions. These approaches then rely on third parties to provide services of contract negotiation, business partner discovery, runtime contract
monitoring and enforcement, etc. Therefore electronic contracts are mostly some formal representations of traditional paper based contracts that depend on various services provided by third parties.

Adaptive software system contracts not only provide description of allowed interactions but also manage behavior and relationships of two or more loosely coupled software components with respect to each other though these later capabilities vary between different frameworks. The software components managed by these contracts are actually various service providers and service consumers that are part of the software architecture. Therefore contracts in adaptive software frameworks are rich entities capable of performing (to varying degrees) various operations like monitoring and enforcement of interactions between contracted parties by themselves. Contracts inside ROAD framework are also dynamic entities that manage their own life cycle while monitoring and regulating the interactions between the role players. Like other adaptive software architectures ROAD contracts also do not rely on any third party services. But all approaches under this category only give high level description of contracts and do not provide details about the design and implementation of contracts used in the frameworks.

2.3 Summary

In this chapter we provided background to this study with the introduction to the types of contracts used in the field of software engineering. We discussed the characteristics of various electronic contracts used for B2B collaborations and analyzed contract life cycle management in three electronic contract specifications in detail. We also discussed contracts used in adaptive software systems and provided an overview of three contract based adaptive software systems. We then compared electronic contracts, adaptive software system contracts and ROAD contracts.

The Table 2-2 below summarizes various characteristics of electronic contracts reviewed in this chapter. We evaluate various designs against the list of features we have developed in ROAD contracts. We look at the description of the life cycle of the contract, formal representation of contract terms to facilitate monitoring of contract breaches,
monitoring execution of contract terms to validate performance, relationship between breaches in contract terms and life cycle of the contract, we also look at handling of contract breaches in general as some frameworks discuss handling of violations but do not address the effect of violations on contract life cycle. Another desired feature is a flexible compliance model for dealing with contract breaches. Characteristics marked with a tick ✓ are clearly addressed, or can be clearly inferred, from the work reviewed. Characteristics marked with a cross x are not addressed at all by the research work. Characteristics marked with a tilde ~ are partially addressed but complete picture is missing. We have only included electronic contracts in this table as literature on adaptive software system contracts does not provide a clear description on provision of these features as the frameworks are more focused on achieving adaptation and performing reconfiguration operations through contracts.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description of life cycle</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>~</td>
<td>✓</td>
</tr>
<tr>
<td>Representation of contract terms</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>~</td>
<td>✓</td>
</tr>
<tr>
<td>Monitoring contracts at run-time</td>
<td>x</td>
<td>✓</td>
<td>~</td>
<td>~</td>
<td>x</td>
<td>~</td>
<td>✓</td>
</tr>
<tr>
<td>Establishing effect of breaches on contract external state</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>~</td>
<td>✓</td>
</tr>
<tr>
<td>Handling contract breaches</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compliance model for breaches</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2-2: Summary of characteristics of Electronic contracts
3. Architecture of ROAD Contracts

In this chapter we examine the architecture of ROAD contracts in detail. Our approach builds on top of the design presented in [1] by Colman. Our design provides formal expression of terms, obligations and general clauses and illustrates how these layers relate to each other. Our approach applies the concepts of deontic logic and temporal constraints for expression of terms and obligations of contracts. We also illustrate how performance measurement in ROAD contracts is facilitated by the layers within ROAD contract.

As stated before, ROAD contracts not only specify the obligations and permissions granted to contracted parties but also act as a runtime entity that monitors and to some degree enforces those obligations. ROAD contracts encapsulate information about the contracted parties, allowed functional interactions (Terms) between the parties and non-functional (performance) requirements of those interactions. In addition, ROAD contracts also specify an authorized sequence in which interactions between the contracted parties can occur.

In order to realize such rich functionality within a contract, an architectural approach to contract design was undertaken. Functionality of dynamic and rich contracts is logically divided into three layers as follows: The top layer consists of general clauses that govern the life cycle of contracts. In the middle conversation clauses express the interaction protocol to be followed by the contracted parties. The bottom layer encapsulates functional and non-functional requirements of contracts inside the terms. Each term holds its related non-functional or performance requirements inside one or more obligations. Execution of terms is constantly monitored to measure the actual performance of terms against required performance. One aspect of monitoring of term executions involves checking the terms against conversation clauses. Each term is part of a pre-determined conversation and has a protocol that must be followed during execution and this checking is performed by conversation clauses. Conversation Clauses are not part of this study and
are not discussed any further. For details about conversation clauses please refer to Pham et. al. [2]. We discuss other layers of ROAD Contract in the following sections. The figure 3-1 below presents the UML diagram of ROAD contracts.

![Figure 3-1: UML diagram of ROAD Contracts](image)

### 3.1 General Clauses

General Clauses define the rules that govern the contract as a whole and can be seen as management layer in a contract. General clauses specify conditions relating to the commencement, continuation and termination of a contract and manage the entire life cycle of ROAD contract. Therefore general clauses control the external representation of internal state of the contract at any point in time. ROAD contract life cycle states can be grouped into design time and run time states. At runtime ROAD contracts require explicit
representation of its current state as each runtime state in its life cycle represents state of fulfillment of terms of the contract at any particular point. This state is evaluated from the state fulfillment of performance requirements of a contract. In this study we establish the effect of the breaches in the terms of a contract (when actual performance of contracted party is lower than the agreed performance levels) on the life cycle of the contract and the role played by general clauses in managing state transitions. We present a generalized life cycle model for ROAD contracts where transition in states is triggered by violations in contract terms and validations of prior violations. Here we are primarily concerned with the runtime states where contract is monitored for performance. The life cycle of ROAD Contracts is presented in the next section with detailed discussion of each state.

3.1.1 Life Cycle of ROAD Contracts

A ROAD contract progresses through various states after it is created and negotiated between the contracted parties. Contract is in “Incipient” state after its negotiation is finalized by the contracted parties and before it is activated by first invocation of a contract term by either party. As parties execute different terms, ROAD contract monitors each execution and contract transforms to different states depending on the state of fulfillment of terms at any point. The violations to the agreed contract terms by a contracted party place contract in a breached or non-complaint state. A serious breach in the terms may put contract into a suspended state and some extreme breach may even terminate the contract. Contract is constantly monitored at runtime and performance evaluation is escalated from lower layers of the contract, i.e. from obligations to terms and then to general clauses. General clauses then further calculate the actions to be taken in response to performance results received from lower layers and transform contract to different states as a result of those calculations. The figure 3-2 below shows various layers inside ROAD Contracts and escalation of performance results in the form of compliance levels from lower to higher layers. We discuss each state in the contract life cycle in the following sections.
3.1.1.1 Contract Preparation
During preparation state contract is created by ROAD Composite organizer and various terms and clauses are specified. After setting values for various terms and conditions of the contract, organizer creates role definitions and looks for players to fulfill those roles. Negotiation process is started with potential players and on successful negotiation of contract terms between the two players contract is signed.

3.1.1.2 Incipient
As contract negotiation is finalized between the two parties, both parties sign the contract on agreed terms and clauses. This can be achieved through an electronic signature or some other means (not part of this study). The contract at this stage is signed and but is not active as yet. The contract can be activated either by a timer event signifying start date/time of the contract or by the first term invocation by a contracted party. The contract is said to be in incipient state after the two parties have signed the contract but before it is activated. From this state the contract can progress to active (complying) state or active (breached) state as parties start executing terms depending on the performance of the executed term. If a party pulls out of the contract at this stage, contract ends abnormally and proceeds to terminated state before it is finalized by composite organizer.
3.1.1.3 Active (Complying)
As discussed above ROAD contract enters this state either with a timer event or after first term invocation by a contracted party. In active state parties execute contract terms and actual performance of the terms is compared with required performance as stored inside obligations. The results of this evaluation are then passed to the layers at the higher level, i.e. to the terms and general clauses. In our approach we use event based architecture for managing compliance levels and runtime states of the contract. (The application of ECA rules to ROAD Contracts is discussed in the next chapter) As general clauses receive the compliance levels of the terms with respect to its obligations, general clauses establish the effect of those compliance levels on the external state of the contract. Incase of a breach, general clauses may transform the external state of the contract to another state depending on the severity of breach. For example a low level breach may result in taking contracted to a breached state but contract still stays active. An additional action by general clauses may be notifying composite organizer about the occurrence of the breach by sending a message. The consequences of higher level breach could be much severe resulting in suspension or termination. General Clauses may also specify any recovery actions to be performed due to the breach as established from the evaluation of compliance levels. The resulting actions for the various levels of breach/compliance are specified by the ROAD designer at the design time as per the requirements of the application that is being built using ROAD Framework.

3.1.1.4 Active (Breached)
This state is very similar to Active (Complying) state and contract execution proceeds as normal. The difference is that a minor breach in the contract terms has occurred and has been reported to the general clauses in the form of compliance level of a term. The general clauses have taken some action including changing contract state to Active (Breached) as specified at design time. At design time ROAD designer specifies the minimum threshold levels for performance of terms before contract falls into breached state. The breached state can be corrected by the contracted party responsible for term violation (also called obligated party) either by improving its performance or paying some penalty as agreed during negotiation of terms. Contract functions in the similar way
as in Active (Complying) state but again the exact behavior can vary depending on application requirements as specified by the application designer at design time. The number of Active (Breached) states that can be specified by the ROAD designer at design time can vary as well. We discuss multiple breached states with the contract compliance model introduced in the next chapter.

3.1.1.5 Suspended
Contract is transformed to this state from Active (Complying) or Active (Breached) state on occurrence of a serious breach. A point to be reiterated is that the actions resulting from occurrence of breaches are specified by the application designer at design time and depend on requirements of a particular application. Different applications will have different requirements on how a breach should be handled. An application may require that a contract should never be automatically suspended by general clauses and therefore a serious breach should only result in general clauses sending a message to the composite organizer to inform about the breach and wait for actions from the organizer. On the other hand, another application may gain more value with automatic suspension of contract on occurrence of some serious violation in contract terms without waiting for organizer to perform some action. The contract can then be reactivated by a corrective action taken by the obligated party. This type of behavior provides more automatic handling of breaches as there is less dependency on composite organizer. Further ROAD composite organizer is the ultimate administrator of the composite and can also execute a transition of state if required.

3.1.1.6 Terminated
The termination of ROAD contract implies ending of contract in an abnormal fashion mostly as a result of some very serious breach. The contract may progress to this state from Active (Complying), Active (Breached), Suspended or Incipient. This can be executed by the general clauses or by the composite organizer again as specified at design time. Most applications will not require termination of contract by general clauses and would prefer a human intervention as automatic execution of such drastic action will be
an overkill. A terminated contract cannot be re-enacted and a new instance must be started incase parties wish to reform the alliance.

3.1.1.7 Finalized

Contract proceeds to finalized state on normal completion of contracted period or from a terminated state. In final state contract is destroyed and important information is persisted to a data store by the composite organizer.

The figure 3-3 below illustrates the life cycle of ROAD Contracts divided into design time and run time states as discussed above.

3.2 Terms

The terms of a ROAD contract are clauses that specify the allowed atomic functional interactions between the contracted parties. Unlike conventional paper contracts that may
have some terms implicitly implied based on natural understanding and local laws, ROAD contracts require explicit specification of all allowed functional interactions. Explicit specification is essential in order to achieve the automated contract enforcement at runtime when the actual interactions between the contracted parties need to be evaluated against allowed interactions to establish if the contracted parties are complying with their contractual requirements. In the domain of electronic contracts it is widely accepted [3, 5, 6, 7, 8] that in order to achieve automated monitoring and enforcement of electronic contracts, contract terms must be defined using some formal specifications. The formal modeling of contract terms must not only capture deontic constraints (discussed below) associated with contract terms but also temporal constraints (discussed in section 3.2.2) attached to those terms (Marjanovic and Milosevic [5]). A few research efforts [3, 5, 6] have modeled terms of electronic contracts by expressing obligations, permissions and prohibitions associated with the interactions of contracted parties. We discuss these interaction types in the next section under Deontic Logic. We then discuss the requirements of temporal constraints for expression terms that specify obligations of a contracted party.

### 3.2.1 Deontic Logic

Deontic logic is a field of logic that when applied to electronic contracts defines obligations, permissions and prohibitions associated with contracted parties with respect to each other. Marjanovic and Milosevic in [5] define obligation as a prescription that particular behavior must occur and is fulfilled by occurrence of prescribed behavior. Permission is a prescription that a particular behavior is allowed to occur and prohibition is defined as proscription that a particular behavior must not occur. Prohibition can also be seen as negative obligation. Deontic logic is a common approach taken to formally express terms of electronic contracts. Such expressions facilitate runtime monitoring and enforcement of contract terms where breaches in terms of a contract must be identified immediately so that corrective action can be taken by the defaulted party or contract manager. For example library-book vendor contract specifies permission as: *Library can place orders for books from book vendor.* Obligation as: *Book Vendor is obliged to acknowledge the receipt of orders.* If library does not require any new books and hence
no book order is placed, this does not lead to any breach in terms of a contract or it is not a non-normative behavior by the library. On the other hand if book vendor forgets to send an acknowledgement for the received book order, this will lead to breach in terms of the contract as book vendor has not fulfilled its obligation or its behavior is non-normative as per terms of the contract. Kent et. al. [25] emphasizes that use of deontic logic in modeling contract clauses helps in directly distinguishing normative behavior from non-normative behavior. This logic can be also applied to ROAD contract enforcement that needs to differentiate contract breaches (non-normative behavior) by contracted parties from contract fulfillment (normative behavior) by specifying terms as permissions or obligations of contracted parties. Though an obligation in deontic logic specifies the action or behavior that must be performed by a party, it does not impose any temporal constraint over the occurrence of that action. In the next section we discuss the requirement of imposing temporal constraints over obligations of contracted parties.

3.2.2 Temporal Constraints

Marjanovic and Milosevic in [5] argue that modeling of temporal constraints is critical in order to prepare and verify electronic contracts. The paper states that every obligation must be bound by a temporal constraint or it is impossible to enforce it. On the other hand permissions are not temporally bounded as there is no requirement for a permission to be fulfilled. For example in a contract between a library and book vendor, library has a permission to send requests for books to book vendor. If the library did not place an order as no books were required, this will not result in any kind of breach. Whereas after an order is received by the book vendor, it is obliged to deliver the order in 5 days or else a contract breach will occur. If we do not bind the obligation of book vendor to deliver books with a temporal constraint (5 days), it is not possible to enforce the obligation as book vendor can take for ever to deliver books to the library. Without a temporal constraint a book vendor can send books delivery after 1 year and still satisfy its contractual obligation but will provide little value to the library. Therefore when we model a term as an obligation of a contracted party we also need to specify the temporal constraint that binds the obligation. Authors in [5] differentiate three types of temporal
constraints that can be associated with a term that specifies an obligation. The three types of temporal constraints that can be imposed on obligations are:

**Absolute Time** – Absolute time specifies a particular time point. E.g. Books must be supplied on or before 25/12/2008. Hence this temporal constraint specifies before and after relations.

**Relative Time** – This concept models time independently from any particular time point as it measures time relative to some time or event. E.g. Books must be supplied within 5 days of order received where receiving of order is an event.

**Recurring Time** – This concept is used to describe events that need to occur regularly starting from a particular time point and until some final time point. Hence this concept combines previous concepts of absolute and relative times. E.g. Payments must be made by the library for books delivered by the book vendor within 10 days of receiving each delivery starting from 25/12/2008 to 24/12/2009.

The ROAD contracts contain explicit definitions of all valid interactions therefore prohibitions are implicitly imposed where attempted interaction by a party is not specified. A term of a ROAD Contract by itself expresses permissions granted to a contracted party. We propose that obligations on terms can be specified by attaching a temporal constraint to the term of the contract. Our approach uses XML for formal representation of terms of the contracts encapsulating deontic nature and temporal constraints of the terms. In the following sections we illustrate how the concepts of deontic logic and temporal constraints can be applied to terms of ROAD contracts.

### 3.2.3 Terms – Granting Permissions

As deontic logic is a widely accepted method for formal representation of terms of electronic contracts we model ROAD Contract terms based on the concepts of permission and obligation in deontic logic. In ROAD Framework every contract term must be explicitly specified and therefore it imposes deontic concept of prohibition implicitly as
execution of any non-defined term is prohibited. Applying deontic logic we can define permission in ROAD contracts as an allowance to a party to perform certain action but the contracted party is not obliged to carry out that business action. The terms of ROAD contracts represent permissions granted to contracted parties. For example library-book vendor contract grants permission to library to place orders but the library is not obliged to do so if books are not required. We must emphasize the point that when we grant permission to a party to execute a term e.g. place an order, term is not required to be bound by a temporal constraint as measurement of its fulfillment is not required at any point in time. XML representation of permission term is in figure 3-4 below. Ignore the ECA section at this stage as we introduce it in next chapter.

![XML representation of a term without any associated obligations.](image)

**Figure 3-4: XML representation of a term without any associated obligations.**

### 3.2.3.1 Expressing constraints on permissions

While terms of ROAD Contracts express permissions granted to contracted parties, there can still be some constraints on those permissions that must be met if the party chooses to execute such a term. For example a contract term may state: *Library is allowed to place orders for books to the book-vendor but an order must contain at least 5 books.* According to this contract term though library is not obliged to place an order but if it chooses to do so, the number of ordered books must be at least 5. This type of constraint can be expressed as a non-temporal obligation of a contracted party and can be enforced
by the contract at runtime as it intercepts term execution messages. We elaborate on non-temporal obligations in the following sections.

3.2.4 Terms – Enforcing Obligations

ROAD contract terms express permissions granted to a contracted party to perform some actions as discussed in the previous section. In order to incorporate the other aspect of deontic logic i.e. obligation into the ROAD contract terms, we need to bind the term with a temporal constraint or an event; e.g. book-vendor to delivery books in 5 days. This term bound by a temporal constraint of 5 days specifies an obligation of the book vendor. Such temporal constraint can be seen as a non-functional requirement of the term of the contract. In addition to temporal constraints, a contract term can also have other non-functional requirements that refer to some other non-temporal performance requirements. In ROAD contracts the performance requirements and temporal constraints of a term are expressed as obligations related to the term. We discuss obligations of ROAD Contract terms in the following section. XML representation of term that references an obligation is given in figure 3-5 below.

Note: We are not addressing specification of obligations that are bound by occurrence of external events. For example a term of a contract may state that: book-vendor to deliver books for certain discipline when books are released by the publisher. This type of obligation enforcement requires monitoring of external events and we are only modeling events represented by messages passing through a ROAD composite in this study.
3.3 Obligations

Obligations in ROAD contracts are performance requirements on terms of the contract. The performance requirements are also sometimes referred to as non-functional or quality of service requirements of a term. A term can have zero or more obligations attached to it. Obligations not only express performance requirements on terms but at runtime also measure actual performance against required performance using one specified metric. Therefore if a ROAD contract term has multiple performance requirements associated with it, it must have that many obligations to evaluate the fulfillment of each performance requirement. Also each obligation measures the performance according to the specified metric for individual performance requirement.

Figure 3-5: XML representation of term with associated obligations.
3.3.1 Types of Obligations

The obligations express the performance requirements of the terms that by themselves only express permissions granted to a contracted party. Obligations express and measure different types of performance requirements of a term. We broadly categorize the performance requirements of terms as temporal or non-temporal. We do this categorization in order to emphasize the importance of temporal constraints for enforcing terms. Following from the discussion of deontic logic and temporal constraints we note that in order to enforce a term as an obligation of a contracted party it must be associated with at least one temporal obligation but may further have one or more non-temporal obligations. The non-temporal obligations associated with it express its non-temporal performance requirements. A term can have only non-temporal obligations but then the term only implies permission granted to a contracted party where the non-temporal obligations are evaluated only if the term is executed and the term by itself is not enforceable. The term that has more than one obligation also contains some logic that specifies the order in which the obligations should be invoked for evaluation. We present the temporal and non-temporal types of obligations in the following sections.

3.3.1.1 Temporal Obligations

Temporal obligations express temporal constraints associated with the execution of terms of ROAD Contracts. A temporal obligation expresses a temporal constraint and measures the performance of a term execution against that constraint. The required performance of a temporal obligation is expressed as a temporal constraint that specifies a deadline for fulfillment of obligation. Temporal constraints can be expressed using three types of temporal concepts discussed in the section 3.2.2. Using these concepts we can specify temporal constraints that express an absolute time point, a relative time with respect to some event/action or a recurring time. When expressing temporal constraint with an absolute time value we specify a time point with an absolute date/time value like 11:00 A.M. on 25th Dec, 2008. Relative time constraint expresses when an action must be taken relative to the beginning or end of an event/action. Relative time can be specified as 5 days after occurrence of order confirmation message event. The validity period (Term is valid between certain dates) of a term of a ROAD contract is specified with date/time
values in the term itself and term is valid only between those time points. The validity period of a term can be the same as the contract or less but must be contained within the validity period of contract. Therefore in a ROAD contract a recurring time constraint can be expressed by specifying an event and a relative time value bound by the validity period of the term. Such a constraint will imply that an action must be taken after a relative time value after occurrence of the specified event and every time that event occurs or at some constant time intervals within the validity period of the term. Therefore recurring time constraint uses the concepts from absolute and relative constraints but a recurring constraint is valid for the entire time period for which a term is valid.

3.3.1.2 Non-Temporal Obligations

Non-temporal obligations express non-temporal performance requirements of terms. For example a ROAD contract term states: Book-vendor to deliver books to the library. The temporal obligation of this term states: books must be delivered within 5 days of order. Therefore this term has one temporal obligation that states the timeframe within which the books must be delivered. Temporal obligation will evaluate compliance to 5 day delivery deadline by checking if the book delivery has arrived by the end of 5 days. Other obligations of the same term can specify the conditions in which the books must be delivered (not torn, wet or damaged) and that delivered quantities must exactly match the ordered quantities. These obligations also need to be evaluated to establish compliance (and level of compliance introduced in next chapter) to the agreed contractual requirements by the contracted parties. Therefore we express quality requirements of terms as non-temporal obligations of contracted parties as the level of fulfillment of these requirements results in establishing compliance levels with respect to such requirements as illustrated by the example below.

A term of library-book vendor contract can have non-temporal obligations that specify the quality of books (not torn, wet or damaged) and quantity requirements (received quantity equals ordered quantity) of the delivery. The evaluation of these obligations also requires another entity called performance providers. The performance providers provide the actual performance values and in the same metric as specified by the obligation.
(Performance providers are introduced at the end of section 3.3.2.9.1) For example while evaluating compliance to accurate quantity (delivered quantity is equal to required quantity); actual performance is received by an obligation from performance provider as a number (double) specifying percentage of accuracy of delivery. The obligation compares this value with the number (double) stored in the obligation specifying the required performance (percentage accuracy) of the delivery. The result of this evaluation establishes compliance with respect to the ordered quantities. In this example if actual performance is only 80% while the required performance was 100% (i.e. delivered quantities had to exactly match ordered quantities), the book vendor is in breach of the contract. Another point to be noted is that these obligations are only evaluated once and after the delivery is received. It will not make any sense to evaluate the quality of books delivered before the delivery actually arrives at the library, so the term will only invoke the book quality obligation on successful completion of an obligation that confirms arrival of delivery.

When a term is associated with only a non-temporal obligation, the term expresses a permission granted to the contracted party and non-temporal obligation is only enforced if the party chooses to execute its permission. Therefore in this case, the obligation is invoked by the term only when general clauses invoke the term to retrieve the compliance level of its obligation. For example a term states that: Library can place orders for books from the book-vendor. This term states permission granted to the library. But this term has a non-temporal obligation stating that the number of books ordered must not be less than 5. This obligation is only evaluated after an order is placed and contract intercepts a message that contains the order.

3.3.2 Aspects of obligations

Obligations express performance requirements on contract terms and at runtime measure the actual performance against required performance to evaluate compliance (or level of compliance) to the required performance. Therefore an obligation associated with a term facilitates runtime enforcement of agreed performance levels as negotiated between the contracted parties at design time. Obligations further contain some attributes that
facilitate compliance evaluation and escalation at runtime by comparing actual against required performance. We discuss various attributes of obligations and then present formal expression of obligations using XML in the following sections. An Obligation contains following attributes:

### 3.3.2.1 Obligation type
The obligation_type attribute of an obligation specifies the temporal or non-temporal type of an obligation. This distinction is necessary for application designer to ensure every term to be enforced has been associated with a temporal obligation. The temporal obligations may also be required to be evaluated multiple times depending on the number of compliance levels required by an application whereas non-temporal obligations are evaluated once regardless of the number of compliance levels. (Compliance levels are introduced in the next chapter)

### 3.3.2.2 Domain variable
The domain variable attribute of obligation specifies the name of the domain specific variable whose performance is evaluated by the obligations. For example in library-book vendor contract the actual value of delivery_quality domain variable is checked against its required value stored in obligation to evaluate compliance at runtime. The temporal obligation discussed above has domain variable called delivery_deadline. At runtime, compliance is evaluated by checking if the delivery has been made by the temporal value held in delivery_deadline variable.

### 3.3.2.3 Required performance value of an obligation
As discussed above, an obligation expresses a performance requirement of a term. The measurable value of the performance requirement of the term is expressed in required_performance attribute of the obligation. This attribute contains a literal value that expresses the required value of the domain variable attribute at runtime and this is the value that is compared with the actual performance value to evaluate compliance to an obligation. This value can be a number, Boolean or a string that can be compared with the actual performance at runtime.
3.3.2.4 Metric for performance evaluation

The required performance attribute only specifies a literal value but we also need to express a metric used to measure that value. “Metric” attribute is used to express the metric of the value specified in required performance attribute. For example ROAD contract term discussed in the previous sections states: *Books to be delivered by the book vendor.* The obligation associated with this term states: *books to be delivered within 5 days of order.* The obligation associated with this term holds value “delivery_deadline” inside domain variable attribute, “5” in required performance attribute and value “days” in metric attribute. Any metric can be used to express the required performance of an obligation but it must match the metric in which the actual performance measurement of execution of that obligation at runtime is received by the contract. Only then an accurate evaluation of actual against required performance can be performed.

The metric for measurement of required performance of an obligation must be expressed in enough details in order to allow correct and accurate evaluation. For example when time is being calculated it must be precisely specified if it is being measured in second, minutes, hours, days, etc and actual performance must be received in precisely the same unit. The metric can be temporal, a number, a boolean value or a string enumeration. In this study we are only looking at simple types as metrics of an obligation and leave the expression of complex metric types in obligations for future work. A temporal metric can be an absolute time or a relative time. An absolute time metric specifies that required performance value can be an absolute date/time value e.g. 25/12/08. A relative time metric can be specified as a time period (e.g. days, minutes, seconds etc) that is measured after occurrence of some event. We can specify this event as a parameter of the metric. When a string enumeration is expressed as a metric, actual performance value must be one of the strings in the enumeration specifying the required performance values. The simplest metric is of type Boolean. An obligation with metric of type Boolean receives a Boolean value as actual performance of an obligation and evaluates compliance by comparing two Boolean values.
3.3.2.5 Comparator
The comparator attribute of an obligation specifies the comparator used to compare actual
performance with required performance at runtime. For example “\(\geq\)” can be used as a
comparator for comparing actual against required performance when comparing numbers.
Some other possible comparators are: “\(=\)”, “\(<\)”, “\(\leq\)”.

3.3.2.6 Actual performance value
Obligations receive actual performance values of domain variables from performance
providers as terms are executed by the contracted parties. Further performance providers
provide these values in the metric required by the obligations. Compliance is evaluated
by comparing this value with the required performance values using the comparator.

3.3.2.7 Activity completion variable
Activity completion variable is a Boolean variable and only applicable to temporal
obligations as these obligations need to check the completion of the underlying activity
associated with the term with respect to the temporal constraint expressed in the
obligation. Temporal obligations only need to read the value of this variable as its value
is set by a clerk who updates the status of the underlying activity as it is completed. For
example in the term: *Books to be delivered by the book vendor to the library*, the delivery
deadline temporal obligation states that: *the delivery is to be made within 5 days of order.*
Now when the temporal obligation of this term evaluates compliance to the delivery
deadline it needs to know the completion status of the underlying activity, i.e. the
delivery of books in order to evaluate if the delivery was made before the deadline. The
default value of activity completion variable is *false* and it is updated to value *true* by the
clerk in the library who updates the delivery status in the system as the delivery is
received. If the activity completion variable has value *false* after 5 days, the book vendor
has breached the contract as the delivery has not been made by the deadline. This variable
is not required in a non-temporal obligation as non-temporal obligations only check the
actual performance values received from the performance providers with the required
performance and do not need to check the completion of underlying activity in order to
evaluate compliance. This is because the non-temporal obligations are always invoked by
their parent terms after compliance evaluation of temporal obligations confirms the completion of underlying activity. If a term only contains a non-temporal obligation, then the obligation is triggered by the term only on a message interception that confirms execution of the term.

### 3.3.2.8 Status of an Obligation

Status of an obligation can have four values as shown in figure 3-6 below.

![Figure 3-6: Status of an Obligation](image)

The status of an obligation has value “NotApplicable” before a term execution begins. This is natural as before a term is executed by a contracted party the status of all obligations associated with that term is not applicable. This status changes to “Pending” with the execution of the term. Pending status implies that a term has been executed but the evaluation of its obligations is pending. As an evaluation is performed and compliance of an obligation is calculated, the status of obligation changes to checked or terminated. Terminated status implies that a compliance of an obligation has been calculated and the calculated compliance is final requiring no further evaluation. The checked status of an obligation only applies to temporal obligations that require multiple checks corresponding to the array of threshold values expressed in compliance functions. Compliance functions are basically invoked by obligations to evaluate level of compliance when the actual performance fails to satisfy the required performance. (Compliance functions are introduced in the next chapter). For now we can assume that the compliance functions contain an array of threshold values that correspond to different levels of compliance. Incase of a temporal obligation that has a compliance function associated with it, multiple evaluations of the obligation is required incase the underlying
activity is not completed by the obligated party when compliance is evaluated. The completion status of this underlying activity is accessed by the obligation by checking the value of the activity completion variable discussed earlier. The status of a temporal obligation can be changed from “checked” to “terminated” either with the completion of the underlying activity (when activity completion variable returns true) or when the compliance function returns the lowest compliance level corresponding to the lowest set threshold value. In both the cases the established compliance level is final and further evaluations are not required. If a temporal obligation has no associated compliance function, the very first evaluation provides the final compliance result of the obligation and the status of the obligation proceeds from pending to terminated.

Analyzing the library-book vendor contract term again, the status of obligation that checks delivery compliance to 5 day deadline is “NotApplicable” before an order is placed. The status changes to “Pending” after an order is placed and till the 5 day period when the activity completion variable is checked by the temporal obligation to verify if the delivery has been received. If the delivery has been received after first check the resulting compliance evaluation is final and the status changes to terminated as the underlying activity (book delivery) has been completed. If the book-vendor did not deliver books after 5 days but the obligation has no associated compliance function, the status of obligation changes to terminated again. This is because though the activity completion variable has value false implying that the underlying activity is incomplete but the final compliance result has been established as there are no further requirements of compliance evaluation. In this scenario there are no levels of compliance after you breach the required performance and first evaluation provides the final result i.e. compliance or non-compliance. But if the obligation has an associated compliance function that holds an array of threshold values corresponding to different compliance levels, the compliance evaluation is required further if the delivery is not received in 5 days. In this case after the first check, though we can confirm that the book-vendor has not complied with the required performance by not delivering the books within 5 day deadline but we still need to perform more checks until books are delivered or the lowest compliance level is returned by the compliance function. This is necessary to verify if the
book delivery activity is completed and also to evaluate the extent of deviation of actual performance from required performance. Level of compliance is lower when books are delivered in 15 days than when the books are delivered in 7 days. (Multiple Compliance levels are discussed under the Compliance Function introduced in next chapter). A compliance evaluation that establishes the arrival of delivery at the library also provides the final compliance level of the obligation and the status of obligation changes to terminated. If the books are never delivered, the final compliance result is established with the return of lowest compliance level by the compliance function of the obligation. Therefore the evaluated compliance is final compliance of an obligation when either the activity completion variable returns true value during compliance evaluation or there are no further threshold values to be evaluated in the compliance function.

3.3.2.9 Reference to performance provider
Performance_provider_reference attribute holds reference to the performance provider that is invoked by the obligation to obtain actual performance value at any time. This value can be a URI or object reference.

3.3.2.9.1 Performance Providers
Performance providers fulfill the role of providing actual performance values of a contract term execution at runtime. Performance providers are independent software components that also perform the work of an adapter by providing the actual performance measurement in the metric required by an obligation of a term. The performance providers interact with the contracted parties to provide actual performance values to contracts inside ROAD composite. The design and implementation of performance providers is beyond the scope of this study. The figure 3-7 below illustrates the role of performance providers. Please note that the figure only shows the conceptual view of the interaction between performance providers and contracts inside ROAD composite.
### 3.3.2.10 Current compliance level

Compliance level is calculated by comparing actual performance against required performance by using the specified comparator. The compliance_level attribute holds current compliance level of an obligation at any time. As discussed above temporal obligations may have their compliance levels evaluated a number of times and the values of this attribute may change with each evaluation.

Other attributes of an obligation are obligation ID and parent term reference ID. These are required as each obligation needs to identify itself to the term when it escalates compliance level to the parent term and to actually know the term that the obligation is associated with. An obligation may also have an associated compliance function that is introduced under the compliance model in the next chapter.
3.3.2.11 XML representation of obligations

The following figure 3-8 presents the formal expression of an obligation using XML that evaluates compliance for “DeliveryQuantity” domain variable. Ignore the ECA rule part at this stage as we discuss it in next chapter.

Figure 3-8: Representation of obligation in XML

The figure 3-9 below presents XML schema for obligations that the above XML representation conforms to.
Figure 3-9: Obligation XML Schema
Figure 3-9: Obligation XML Schema (Continued)
3.4 Evaluating Contract Performance

ROAD contracts unlike conventional contracts enable automated handling of contract breaches. As discussed in the section 3.1.1 under the contract life cycle, ROAD contract is constantly monitored at runtime to check if the parties are fulfilling their contractual requirements. Monitoring of ROAD contract implies monitoring the behavior of contracted parties with respect to the required behavior. Monitoring of behavior thus includes monitoring of the execution of contract terms and associated obligations. As each valid contract term is explicitly specified, monitoring of term execution involves checking the presence of that term in the contract and any protocol that needs to be followed in executing that term. This checking is performed by conversation clauses. For details please refer to Pham et. al. [2]. Any unspecified term is prohibited and attempt at its execution rejected straightaway. Hence the actual monitoring of execution of a term is really the evaluation of state of fulfillment of performance requirements of that term, i.e. the obligations associated with the terms.

In this chapter so far we discussed various components within the architecture of a ROAD Contract except for conversation clauses that are covered by Pham et. al. in [2]. We will now demonstrate how the various components discussed above are used to evaluate contract performance at runtime. In the following sections we briefly discuss how compliance with performance requirements of a contract term is calculated by obligations and how it is escalated to the general clauses. We elaborate on the details after introduction to compliance model in next chapter.

3.4.1 Evaluating compliance in an obligation

We adopted ECA rules for compliance evaluation and escalation from obligations to general clauses in ROAD contracts. We discuss the application of ECA rules to ROAD contracts for compliance evaluation in next chapter. For now we can assume that the compliance evaluation in obligations is triggered by some events. After being invoked with an event at runtime an obligation in turn invokes its performance provider to get the actual performance value of the variable identified with the domain variable name in the obligation. Performance provider returns the actual performance value to the obligation in
the same metric as specified in the obligation. The obligation then compares both actual and required performance values using the comparator in comparator attribute to evaluate if the actual performance is greater than (or better than) or equal to the required performance. If it is, the obligation returns compliance level “1” that implies full compliance with performance requirement of the term of the contract. Otherwise compliance level “0” is returned implying that the obligation has been breached as actual level of performance is lower than the required level.

3.4.2 Compliance escalation to the term
After evaluation of compliance is complete, the obligation escalates its compliance result to the term by triggering an event on the term. Now if the term contains more than one obligation it may need to get the results of compliance evaluations of all of those obligations before escalating any evaluation results to the general clauses. As discussed earlier when a term contains multiple obligations, evaluation of non-temporal (quality based) obligations follows evaluation of temporal obligations. Also every term that has more than one obligation contains some logic that is applied to formulate any relationship among obligations and any sequence that needs to be followed for invocation of obligations.

3.4.3 Compliance escalation to the general clauses
Terms escalate compliance levels of each of its obligations to general clauses. General clauses contain the rules that evaluate the effect of compliance level of an obligation of a term on the contract as a whole. General clauses may apply some additional logic and combine compliance results from multiple obligations belonging to different terms to evaluate that effect. General Clauses evaluate the effect of internal state of the contract on its external representation by calculating the individual and combined effect of compliance levels of obligations of terms according to the rules established by the application designer at design time. The figure 3-10 below illustrates escalation of compliance levels from obligations to general clauses.
EVENT (Compliance Notification)

CHECK CONDITIONS
If Term1.Obligation1.ComplianceLevel = 1 AND Term2.Obligation1.ComplianceLevel = 0

TAKE ACTION
Change contract state to “Suspended”

Figure 3-10: Escalation of compliance within the ROAD Contracts
3.5 Summary

This chapter started with the description of architecture of ROAD contracts as proposed in ROAD framework. We then presented our approach and described the roles played by the various layers in the architecture. We explained how general clauses control the life cycle of ROAD contracts. We modeled terms and obligations based on the concepts of deontic logic and temporal constraints. We discussed how terms by themselves only imply permissions granted to contracted parties. We described association of terms with obligations for expressing performance requirements of terms. We categorized obligations into two types, i.e. temporal and non-temporal obligations. We emphasized that in order to enforce contract terms as obligations of contracted parties those terms must be associated with a temporal obligation. We discussed the role of non-temporal obligations for expressing quality related performance requirements of terms. We also presented formal representation of terms and obligations in XML. We concluded this chapter with a brief discussion of how obligations, terms and general clauses evaluate performance of a ROAD contract. We elaborate on the details of contract performance evaluation in the next chapter.
4. A Compliance Model for ROAD Contracts

In this chapter we introduce a compliance model for ROAD contracts that enables ROAD contracts to flexibly handle breaches according to precise application specific requirements. We then present detailed discussion of performance management in ROAD contracts with compliance model and ECA rules. We describe the relationship between compliance model and the life cycle of ROAD contracts and also explain how life cycle is managed between general clauses within contracts and ROAD composite organizer.

Contracts in ROAD Framework facilitate implementation of adaptive behavior in a ROAD Composite. Though the framework views a contract as an entity that controls interactions between two contracted parties but fundamentally a ROAD contract is a software system that enables automated handling of contracts that businesses have traditionally handled manually. Business partners have struggled to enforce contracts manually as contract documents are often voluminous and contain a lot of legal jargon making it hard for businesses to find relevant terms that need to be taken care of at any particular time (Krishna et. al. [26]). ROAD Contracts not only encapsulate information about contracted parties and allowed interactions between the parties but also monitor contract execution and facilitate enforcement of contract terms at runtime. At runtime ROAD contracts monitor term executions and general clauses within contracts take appropriate actions when contracted parties do not comply with their obligations. But a real world business application still has more requirements that must be fulfilled before a ROAD Contract can be used to implement relationships between business partners.

A real world business application requires an efficient and flexible approach towards handling of contract breaches. For example let us assume that Swinburne University sets up a ROAD contract with a book vendor to supply books to all its libraries in all campuses. Now if due to some reason a monthly payment as per the terms of the contract is missed by one campus, the book vendor will not be willing to terminate or even temporarily suspend the contract with the University due to loss of business to the book
vendor itself. Book vendor will desire a more moderate approach where a simple message that informs about the breach is sent to ROAD composite organizer or to each contracted party and administrator on the book vendor’s side may even wish to resolve the issue manually after issue is escalated. Whereas if monthly payments are not made by all seven campuses of the University for the same month, the book vendor will be demanding a payment before accepting any more orders and may be ready to temporarily suspend its contract with the University. Therefore a real world application needs flexibility in dealing with breaches. In the traditional paper based contracts this flexibility existed as all contract breaches were raised manually by humans and action was taken according to the seriousness of breaches as per agreed contract terms and conditions. A ROAD Contract also needs to provide such flexibility where a breach can be handled according to its seriousness but without sacrificing any automation. The requirement is a mechanism that enables handling different breaches according to precise business requirements. We introduce compliance model in ROAD contracts as a mechanism for dealing with different types of breaches in the following section.

4.1 Compliance Model

In order to provide flexibility in dealing with contract breaches, we introduce compliance model in ROAD contracts. The compliance model provides flexibility by enabling application designers using ROAD framework to establish multiple levels of compliance inside a contract where each level corresponds to the degree of deviation of actual performance from required performance. The higher the deviation of actual from required performance, the higher the level of breach or lower the level of compliance. This enables general clauses to take selective action according to the level of compliance of obligations of its terms. The terms of a contract define an atomic interaction between the contracted parties. Every term or interaction in fact specifies some business function or requirement and carries different value for the business. Hence the significance of a breach or compliance also depends on the importance of breached term. With a compliance model contract designer can specify different levels of breaches/compliance for different terms of the contract according to the requirements of an individual application.
Depending on the requirements of the application, contracts can have multiple compliance levels that enable identification of both, the significance of a breached obligation of a term to the contracted parties and degree of deviation of actual from required performance. Consequently each level results in different set of recovery actions or events triggered by the contract on behalf of contracted parties. Such actions are taken by general clauses depending on the compliance results of obligations and any other conditions applicable to the non-complaint term. The number of compliance levels in a contract depends on the requirements of the application and it is not possible to specify a fixed compliance model with fixed number of compliance levels that can satisfy the requirements of all business applications. In order to incorporate the various levels of compliance that can be defined according to individual business requirements, we propose a compliance model that gives contract designers flexibility to design their contracts based on number of compliance levels that best satisfies the requirements of their application and is agreed by the contracted parties. In order to evaluate compliance levels for an obligation of a term we use another entity called compliance function. The following section illustrates the role of compliance function in ROAD contracts.

4.1.1 Compliance Function

As discussed in the chapter 3, the obligations associated with terms express performance requirements on terms and also evaluate compliance at runtime by comparing actual performance of contracted parties with required performance set during contract negotiation. The comparison between actual performance and required performance results in either compliance or non-compliance. In order to provide flexibility in dealing with non-compliance or breaches in contract terms, we need to further evaluate non-compliance with the required performance in order to get the level of non-compliance. We introduce compliance functions in ROAD contracts for evaluating the level of non-compliance after an obligation establishes that current performance results do not comply with required performance level. Therefore a compliance function is invoked by an obligation only after non-compliance with the required performance level has been established. A compliance function receives the current performance (or actual
performance that obligation originally received from performance provider) from the obligation and evaluates level of compliance by comparing the actual performance value with a series of threshold values that correspond to different compliance levels. The figure 4-1 below illustrates invocation of compliance function by an obligation. We elaborate on how compliance function evaluates different compliance levels in the next section.

4.1.2 Compliance Levels

The application designer can specify the number of compliance levels required by an application at design time that best satisfy a particular application’s requirements at runtime. Let us assume that the designer of library-book vendor contract discussed in the previous chapter identifies five levels of compliance in the compliance function when the actual performance fails to satisfy required performance in the delivery deadline obligation. A contract term states that: book-vendor is to deliver books to the library and the temporal obligation associated with this term is: within 5 days of receiving an order. If the delivery is not made in 5 days then contract needs to evaluate the degree of non-compliance by comparing actual performance (when delivery is made) with four threshold values of 7 days, 9 days, 11 days, and 15 days each mapping to a different compliance level in the decreasing order. Let us assign values to various attributes of above obligation. The required performance attribute of the above obligation has a value of 5, the metric is days (temporal-relative) and domain variable name is
delivery_deadline. The compliance function contains four threshold values of 7, 9, 11 and 15 and same comparator as the obligation. At runtime if actual performance fails to satisfy required performance (i.e. books are not delivered in 5 days), the compliance function is invoked by the obligation to evaluate the level of compliance. The compliance function then compares actual performance with its threshold values. Threshold values are minimum performance values for achieving a particular compliance level. The compliance function therefore calculates compliance by comparing the actual performance with the threshold values using the same comparator as obligation. If the value of actual performance is greater than (or better than or equal to) a particular threshold value it results in a compliance level set for that threshold value by the application designer. Please note that comparison depends on the type of comparator used. In general, in order to have n number of compliance levels in an application, the designer needs to specify n-1 threshold values. The following discussion illustrates how a compliance function can be designed with five levels of compliance by specifying four threshold values.

**Level (-1) (Almost Compliant State)** – Please note that the first compliance evaluation is always performed by the obligation and if the actual performance value satisfies the required performance value, the result is full compliance and invocation of compliance function is not required. The compliance function is only invoked when the actual performance fails to satisfy required performance. Compliance function returns compliance level -1 when the actual performance value satisfies (is better than or equal to) the highest value in the threshold values inside compliance function. Level -1 implies compliance result slightly lower than full compliance. In our example if books are delivered between 5 and 7 days, the compliance level -1 is returned by the compliance function to the obligation. The action taken by the general clauses for this compliance level can be persistence of the compliance level but application designer may specify more actions or different set of actions if required.

**Level -2 (Minor Breach)** - The actual performance value is lower than the highest value in the threshold values but higher than second threshold value (threshold values are
specified in the decreasing order). In the above example when books are delivered after 7 days but on or before 9 days the resulting compliance level is -2. Action taken by general clauses as a result of this evaluation could be sending some message to composite organizer or contracted parties. As the breach is still minor any actions resulting from this evaluation should not be very drastic but again the actions can vary according to individual application requirements.

**Level -3 (Major Breach)** – The actual performance value is lower than second threshold but greater than or equal to third threshold value. In our example this level is evaluated when delivery arrives between 9 and 11 days. The designer can specify stronger actions than before by sending message to the higher department in the book-vendor or apply some penalty to the book-vendor. Again it depends on application requirements.

**Level -4 (Serious Breach)** – The actual performance value is now lower than third threshold value but greater than fourth threshold value. When the delivery is made after 11 days but before 15 days it will result in compliance level of -4 in this example. Designer may specify suspension of contract after 11 day period has elapsed and the books have not been delivered. The restoration of the contract now may require delivery to be made as well as book-vendor pay some penalty before suspension is lifted. Again it is up to the designer to specify what actions are to be taken.

**Level -5 (Very Serious Breach)** – The actual performance value is even lower than the fourth and the last threshold value. In our example when books are not delivered on expiry of 15 day period the resulting compliance level is -5. As this is the lowest value in the array of threshold values specified in compliance function for our example, the evaluation of this compliance level provides the final compliance level even though the delivery was not made. The application designer may specify termination of contract when compliance evaluation by the obligation of the above term returns level -5 to the general clauses. The table 4.1 below illustrates the compliance levels returned by the compliance function when compliance evaluation is triggered at each threshold value. For example when compliance level evaluation is triggered after 7 days of order, if the
delivery has been made before that evaluation, the resulting compliance level is -1 otherwise it is -2 and so on for other threshold values. Please note that multiple compliance evaluations are required in this case due to temporal type of the obligation. As discussed in chapter 3 under section 3.3.2.8, a temporal obligation may be evaluated multiple times until the underlying activity (delivery in this example) is completed or the lowest compliance level is returned by the compliance function. The completion of underlying activity is checked by reading the value of activity completion variable (delivery status in this example). Also each compliance evaluation result is escalated to the term by the obligation and then to the general clauses by the term so that general clauses can take any further action specified by the contract designer for the evaluated compliance level.

<table>
<thead>
<tr>
<th>Threshold Values</th>
<th>Delivery Status</th>
<th>Resulting Compliance Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 days</td>
<td>True</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-2</td>
</tr>
<tr>
<td>9 days</td>
<td>True</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-3</td>
</tr>
<tr>
<td>11 days</td>
<td>True</td>
<td>-3</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-4</td>
</tr>
<tr>
<td>15 days</td>
<td>True</td>
<td>-4</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-5</td>
</tr>
</tbody>
</table>

Table 4.1: Compliance levels returned by a compliance function associated with a temporal obligation

When a compliance function is associated with a quality based non-temporal obligation, the compliance evaluation is performed only once. For example table 4.2 below illustrates compliance level evaluation in a compliance function associated with a non-temporal obligation that evaluates compliance level based on accuracy of delivered items with respect to ordered items (checking delivered quantities match ordered quantities). The compliance level is evaluated by calculating the deviation of actual performance from required performance. For example if the checking of delivered items against ordered items evaluates that 90% or more of delivered items match ordered items,
compliance level -1 is returned. If the evaluation result is between 90% and 80% including 80%, the compliance level -2 is returned and so on. If the evaluation result is below 50% the lowest compliance level of -5 is returned. The table 4.2 below shows how different compliance levels map to threshold values. Please note that this type of compliance evaluation is only performed once after the delivery has arrived and unlike temporal obligations the first evaluation returns the final compliance level. Also similar to previous example, this compliance function is only invoked when obligation evaluates that the actual performance has failed to satisfy the required performance (100% match between delivered and ordered quantities).

<table>
<thead>
<tr>
<th>Threshold Values</th>
<th>Resulting Compliance Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>90 % Correct</td>
<td>-1</td>
</tr>
<tr>
<td>80 % Correct</td>
<td>-2</td>
</tr>
<tr>
<td>70 % Correct</td>
<td>-3</td>
</tr>
<tr>
<td>50 % Correct</td>
<td>-4</td>
</tr>
<tr>
<td>&lt;50% Correct</td>
<td>-5</td>
</tr>
</tbody>
</table>

Table 4.2: Compliance level evaluation by a compliance function associated with a non-temporal obligation

4.2 ROAD contract management with compliance model and ECA rules

McCarthy and Dayal in [11] propose an event-condition-action (ECA) rule based management for active database management systems. The proposed ECA rules state - when an event occurs, check condition and execute action if the condition is satisfied. The events can be internal or external to the application and include transaction events, temporal events and application defined notification events. Conditions can be some logical expressions evaluated in conjunction with each other or independently. The actions can be sequence of actions or single action that is taken when the condition is satisfied. We have adopted ECA rules for design and implementation of ROAD contracts.
Constant monitoring of obligations for compliance evaluation will incur a massive overhead to the underlying system managing a ROAD composite that contains many contracts with each contract consisting of number of terms and again possibly multiple obligations associated with each term of the contract. We adopted ECA rules for the design and implementation of contracts due to the above reason and also due to implicit event based nature of ROAD contracts. When events occur, evaluation of some rules is triggered and action is taken based on the outcome of those evaluations. We use ECA rules for triggering evaluation of compliance in obligations and escalation of evaluated compliance levels to terms and to general clauses. We also design general clauses as a set of ECA rules that specify the effect of compliance levels of obligations of terms of the contract on the external state of the contract. The following sections illustrate how ECA rules facilitate implementation of compliance model in ROAD contracts.

4.2.1 ECA Events for ROAD Contracts

The obligations, terms and general clauses inside a ROAD contract use ECA rules at each level in order to evaluate and escalate compliance levels. We discuss the events that trigger rules at each level of the contract hierarchy below followed by discussion of implementation of compliance model with ECA rules.

4.2.1.1 ECA events for Obligations

Obligations can be triggered by two types of events, i.e. timer events and evaluation events. Timer events are triggered by timers and according to the required performance value of a temporal obligation of the term being executed. For example the delivery_deadline obligation discussed in previous sections has required performance value of 5 days. Now compliance evaluation of this obligation should be triggered as soon as the 5 day period expires. If compliance evaluation is triggered before 5 days, we cannot evaluate compliance to this obligation yet as the delivery deadline hasn’t expired yet. The conversation clauses specify the order of execution of terms, for example after the order for books is placed by the library, the next term to be executed is the delivery of books from book vendor to the library and that obligation has one associated temporal obligation stating that the books must be delivered within 5 days of order. As contract
intercepts each message sent between the contracted parties and checks its validity before passing it to the other contracted party, contract also starts the timers for the temporal obligations associated with the next term according to its conversation clauses. In this example as contract intercepts a message and confirms that library has placed an order for books from the book vendor, the contract starts the timer that will invoke the temporal obligation of the next term (order delivery by the book vendor to the library) after 5 days. Therefore after the 5 day period elapses, a timer event triggers the compliance evaluation in the temporal obligation.

An obligation can also be invoked with an evaluation event that is triggered by the parent term of the obligation. A term can force compliance evaluation of its obligations if it gets a compliance evaluation request from the general clauses. Term invokes each of its obligations in the required order.

4.2.1.2 ECA events for Terms
Terms can also be triggered by two types of events, i.e. notification events and evaluation events. After an obligation evaluates its compliance level, it invokes its parent term with a notification event to escalate the evaluated compliance level to the term. Term can also be invoked by an evaluation event by the general clauses when general clauses need to retrieve current compliance levels of obligations of the term. For example if a message that confirms the arrival of books delivery is intercepted by the contract prior to timer event triggered notification of compliance level for the same obligation, general clauses can trigger evaluation events on terms. In this way general clauses can force compliance evaluations of obligations of terms without waiting for a timer event triggered notification. General clauses may also query current compliance levels of obligations of a term by triggering an evaluation event on the term if queried by the ROAD composite organizer.

4.2.1.3 ECA events for General Clauses
Similar to the terms of ROAD contracts, general clauses can be triggered by notification and evaluation events. When a term wants to escalate the compliance levels of its
obligations to the general clauses, it does so by invoking the general clauses with notification event. Evaluation events on general clauses are triggered by the ROAD composite organizer when it wants to query the performance of any term of the contract.

Figure 4-2: ECA events for ROAD Contracts
It follows from the above discussion that evaluation events are triggered by layers at the higher level and on the layers at the lower level within contract and it goes down the hierarchy of the ROAD contract, i.e. from organizer to contract and inside the contract from general clauses, to terms and finally to obligations. On the other hand notification events are triggered when escalating the evaluated compliance levels from lower to higher layers in the contract and then to the composite organizer.

The figure 4-2 above illustrates the events that can be used to trigger obligations, terms and general clauses of a ROAD contract. After being triggered with an event, obligations, terms and general clauses check their specified set of conditions and take further actions according to the outcome of condition evaluations as discussed in the following sections.

4.2.2 Implementing compliance model in obligations using ECA rules

As discussed earlier, an obligation evaluates compliance level and escalates it to its parent term. If the compliance evaluation by an obligation shows non-compliance as the actual performance fails to satisfy the required performance and that obligation has an associated compliance function, the compliance function is then invoked by the obligation. Obligation passes the actual performance value while invoking its compliance function and compliance function returns the compliance level of that value to the obligation. ROAD contracts use events to trigger compliance evaluations in obligations as explained in the previous section. Invocation of an obligation always results in evaluation of compliance and escalation of evaluated compliance to the parent term. The escalation to parent term happens regardless of the result of evaluation as further effects of compliance results are calculated at the higher levels in the hierarchy of ROAD contracts.

The condition part of ECA rules is not relevant to obligations as same action is always taken by obligations on being triggered by an event and no rules need to apply to calculate the required action. The figure 4-3 illustrates compliance evaluation by an obligation in response to events.
4.2.3 Implementing compliance model in terms using ECA rules

Terms can be triggered by two types of events i.e. evaluation and notification events as discussed above. On receiving compliance level of its obligation with a notification event, the term checks some conditions as the action depends on the outcome of those checks. If a term has more than one associated obligations it may need to evaluate related obligations before escalating compliance levels to general clauses. When related obligations are found, the term will trigger evaluation event on those obligations and on receiving compliance levels of all related obligations, a term escalates compliance levels of each of its obligations to the general clauses.

Term may need to evaluate some other obligations based on the compliance result of one obligation. For example if the compliance evaluation of delivery deadline obligation establishes that the delivery has been made by the book vendor as the status of obligation has been changed to “terminated”. When this compliance result is escalated to the term, term will trigger evaluations of quality based non-temporal obligations by triggering
evaluation events on those obligations. On the other hand if the compliance evaluation establishes that the delivery is still pending as the status of obligation is “checked”, the evaluations of quality based obligations is not required but the term needs to escalate the compliance result of delivery deadline obligation to the general clauses where the impact of non compliance on the whole contract is evaluated.

Also on receiving evaluation events from general clauses, terms need to check the order in which obligations must be triggered. For example in the library-book vendor contract, the term states that: books to be delivered by the book vendor to the library when triggered by an evaluation event needs to trigger the obligation that evaluates the compliance to delivery deadline before invoking other quality based obligations.

Figure 4-4: Compliance escalation by Terms using ECA rules
4.2.4 Implementing compliance model in general clauses using ECA rules

General clauses receive compliance levels of obligations of all terms and evaluate the effects of compliance levels on the contract as a whole including its external state. After compliance information is received, general clauses apply various business rules to evaluate actions that need to be taken for those compliance levels. In the example discussed under compliance levels in section 4.1.2, five levels of compliance were specified for the delivery deadline obligation. If general clauses for that contract receive compliance level -4 for the obligation, general clauses may suspend the contract and send messages to contracted parties and composite organizer. On the other hand general clauses may need to evaluate this compliance level in conjunction with some other values and compliance levels of other obligations. For example general clauses may evaluate if any of the days within the delivery period was a public holiday. If that was the case, the suspension may be delayed by that amount of time i.e. the number of days that were public holidays within the delivery period. The resulting actions are domain specific and are specified by the application designer. The figure 4-5 below illustrates evaluation of compliance levels by general clauses to evaluate the impact of compliance levels on the whole contract.

<table>
<thead>
<tr>
<th>EVENTS</th>
<th>CONDITIONS</th>
<th>ACTIONS</th>
</tr>
</thead>
</table>

**Figure 4-5: Evaluating effect of compliance levels by general clauses using ECA rules**
The following figure 4-6 illustrates the escalation of compliance levels through various layers in the hierarchy of ROAD contracts and escalation of contract performance to the ROAD composite organizer.

Figure 4-6: Contract Performance Escalation in a ROAD Composite
4.3 Relationship between Compliance levels and Contract Life Cycle

In chapter 3 we discussed the life cycle of ROAD Contracts. We also discussed the need for explicit representation of current state of the contract. We may just reiterate here that external state of ROAD contract is external representation of its internal condition at any point in time. Unlike the life cycle discussed in chapter 3, the application of compliance model to ROAD contracts provides a variable life cycle to ROAD contracts as each state of the contract can be mapped to compliance level, and number of compliance levels supported by a contract can be specified by the contract designer according to application requirements. Using compliance model we give application designers the flexibility to choose the number of compliance levels for each obligation of the term of the contract that best satisfies their application requirements. We also allow designers to specify the effects of evaluated compliance results of different terms on the contract as a whole inside the general clauses of the contract. With these provisions designers have total control over defining the life cycle of contracts for their individual applications. An application designer chooses the number of compliance levels required in the contract according to the amount of flexibility needed in the application for dealing with the breaches. The greater the flexibility desired in the application, the larger the number of compliance levels required in the contracts.

There are a few basic states in the life cycle of a ROAD contract. A contract is incipient after it is signed and before invocation of first contract term by one of the contracted parties. Contract is active with full compliance at some stage during contract execution when contracted parties are completely fulfilling their contractual requirements. Contract is suspended when contract execution is stopped due to some breach or fault and rectification of that fault can bring the contract back to active state. Contract can also be terminated at some stage if at least one of the contracted parties completely disregards its contractual requirements. Contract must also be finalized after its contractual period is over and if contracted parties choose not to renew the contract. Finalization implies normal completion of contract while termination is abnormal completion due to very
serious non-compliance or fault. We propose that regardless of the domain of application, the application designers will incorporate most of these states in the life cycle of their domain specific contracts. The states that will vary the most in different applications are the ones that represent contract in a breached state where the breach is handled in a different way than suspending or terminating the contract. The discussion in chapter 3 about contract life cycle also included an Active (Breached) state of a contract that represented a state where a minor breach has occurred that doesn’t affect the normal operation of the contract. An application designer can specify some mild actions to be taken when a contract enters such a state to bring the contract back to active (complying) state or apply a penalty to the party responsible for breach.

The aim of the compliance model is to allow application designers to specify a number of breached states as per their application requirements that result from the evaluation of compliance levels by the general clauses. Each state signifies a different extent of overall breach to the contract requirements and designer can specify different actions including recovery actions to be performed on entering each of those states. Different breached states also inform contracted parties about current internal conditions of contracts more precisely. In an application designed for a large organization the designer may specify various breached states and actions on entering each of those states may include informing different departments within the organization about the breach. As the breach gets more and more serious the higher department is informed about the breach every time until the breach is rectified. Also contract execution may proceed as normal in each of the breached states or restrictions could be imposed as the breaches get serious but again depends on requirements of individual applications. The figure 4-7 below illustrates ROAD contract life cycle with n number of active (breached) states.

Please note that the figure below presents incipient, active, suspended, terminated and finalized states as we identified these as basic states that will be required by most applications. Again this is not a requirement but just an example as the application designers are absolutely free to design the ROAD contract states as per their application requirements and their life cycle design can be completely different from our example.
4.4 Self management in ROAD contracts and role played by the ROAD composite organizer

A ROAD contract is capable of managing its complete life cycle. Till now we illustrated how a contract state is determined from the compliance levels of obligations of each term. As discussed general clauses manage contract life cycle by evaluating actions to be taken in response to compliance levels reported from the lower layers of the contract. Compliance levels are calculated at the lower layer (obligations of terms) and escalated to the general clauses by using ECA rules. General clauses are also implemented as a set of ECA rules that are executed to determine actions that should be taken in response to reported compliance levels. Some of the actions that may be taken by general clauses are changing current state of the contract, sending messages to composite organizer and contracted parties, applying penalties to the parties that breach contract terms and basically executing any actions specified at design time.
An application designer specifies the rules to be evaluated by general clauses in response to compliance levels. The designer may provide ROAD contracts with complete control over their life cycle by enabling contracts to execute actions that include state transitions through the complete life cycle of the contract. The level of control given to general clauses depends on the degree of automation required in the application. More the automation required, greater is the control exercised by the general clauses. The amount of control that should be given to the general clauses varies according to the requirements of individual applications. Complete automation is not a desired feature in every application, actually most applications require humans to execute any critical actions. For example in response to very critical breach in terms of the contract, a very low compliance level is received by the general clauses, most applications will still desire that general clauses only send a message to the composite organizer and may be temporarily halt execution but not terminate the contract automatically.

ROAD composite organizer is the ultimate decision maker in the ROAD framework. Most applications will not desire any strong actions be performed automatically by the contract. When some very serious breach occurs general clauses can inform the organizer about the breach and leave it up the composite organizer to make further decisions on required actions. Therefore application designer can specify the level of control given to the general clauses and specify points after which general clauses yield control to composite organizer for any further actions. These points are just some conditions in ECA rules that when evaluated to be true instruct general clauses to inform composite organizer about current situation and wait for response from the organizer before taking any further action. The implementation of ROAD composite designer is outside the scope of this thesis.

4.5 Summary

In this chapter we introduced a compliance model that provides flexibility to ROAD contracts in dealing with violations to contract terms. We discussed the role of compliance function that is invoked by an obligation only after performance evaluation
by obligation results in non-compliance with the required level of performance. On invocation, compliance function evaluates compliance level depending on the degree of deviation of actual performance from required performance and returns the result to the obligation. The compliance model enables contract designers to specify the number of compliance levels on performance requirements of terms of the contract according to requirements of individual applications. Once compliance level of an obligation is calculated, the effect of evaluated compliance level on the state of the contract can be specified in general clauses, again according to the requirements of individual applications. We also discussed application of ECA rules to ROAD contracts that enables escalation of evaluated compliance level from obligations to general clauses within a ROAD contract. We described the life cycle of ROAD contracts that can vary between different applications. Finally we discussed that level of control given to general clauses over the life cycle of contract depends on the level of automation required by an application.
5. ROAD contract test application

In this chapter we describe our test application that implements the Library-Book vendor contract scenario discussed throughout the thesis. The test application was developed as a proof of concept so it implements only a portion of Library-Book vendor contract that illustrates the concepts of compliance levels and relationship between compliance levels and contract life cycle. We start with the discussion of library-book vendor contract with some of its terms and conditions represented in plain English. We then describe how these terms and conditions are expressed in terms, obligations and general clauses of ROAD contract using XML. In the following sections we illustrate implementation of above contract using Java and Drools rule engine. We develop and populate java classes from declaratively defined ROAD contract using JAXB. We demonstrate how compliance levels are evaluated by delivery_deadline obligation associated with a term of the contract and how compliance levels are escalated to the term and general clauses. General clauses are defined as a set of rules in drool engine that are executed in response to varying compliance levels.

5.1 Contract terms and conditions in plain English

For our scenario we assume that Swinburne library sets up a ROAD contract with a book vendor that will supply books to all its campuses. Let us suppose some of the terms and conditions that may exist in this contract. We assume following terms and conditions:

1. Library in each Swinburne campus is allowed to place orders for books where the number of books in each order must be between 5 and 500.
2. On receiving an order, book vendor must deliver the ordered books within 5 working days. If delivery is received after 15 days, penalty of $100.00 will apply to the book vendor. If delivery is not made by 20 days, contract will be suspended until delivery is made and penalty is paid by the book vendor. Contract will be terminated if there is no delivery on expiry of 30 days.
3. On receiving delivery, library must make payment for the delivered items within 10 working days. If payment is made after 30 days, penalty of $100.00 will apply to the library and if payment is not made by 40 days, contract will be suspended until payment and penalty is paid. Contracted will be terminated if payment is not received on expiry of 60 days.

4. Book vendor on receiving payment must forward the receipt for the received payment within 10 working days.

5. Contract identifies Swinburne University as Party A and book vendor as Party B.

6. The contract is valid from and including 01/01/2009 to 31/12/2009 and will expire on 31/12/2009 if not renewed by this date.

7. Business hours implies hours between 9:00 A.M. and 5:00 P.M. Working days implies days from Monday to Friday excluding public holidays.

The above terms will only form part of the whole contract. For example the above contract can have a term where book-vendor is obliged to confirm acceptance or rejection of order within 2 working days of receiving the order. We have modeled this contract with a few terms and conditions in order to keep the scenario simple.

5.2 Representing contract terms and conditions in a ROAD contract using XML

Our approach uses XML for formal representation of terms, obligations and general clauses in ROAD contracts. The test application defines ROAD contract in XML but the ECA rules used at various levels of the contract are expressed in a rules text file (ComplianceRules.drls) that can be read by the drools rule engine and it is not XML based. We identify terms, obligations and compliance functions of ROAD contract from contract terms and conditions presented in the previous section below and define the ECA rules for this application in section 5.4.

Term 1: Swinburne libraries can place orders for books from the book-vendor.
This term has two associated obligations and the table 5-1 below presents the values of various attributes of these obligations.

<table>
<thead>
<tr>
<th>Obligation Attributes</th>
<th>Obligation 1</th>
<th>Obligation 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Each order must contain at least 5 books</td>
<td>Each order must contain less than 500 books</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Non-temporal</td>
<td>Non-temporal</td>
</tr>
<tr>
<td><strong>Domain variable name</strong></td>
<td>Delivery Quantity</td>
<td>Delivery Quantity</td>
</tr>
<tr>
<td><strong>Required performance</strong></td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td><strong>Metric</strong></td>
<td>Number</td>
<td>Number</td>
</tr>
<tr>
<td><strong>Comparator</strong></td>
<td>Greater than equal to</td>
<td>Less than</td>
</tr>
</tbody>
</table>

Table 5-1: Values of attributes of obligations in term 1 of the library-book vendor contract

**Term 2:** *Book vendor is to deliver the ordered books to the library.*

This term again contains two obligations as shown in table 5-2 below.

<table>
<thead>
<tr>
<th>Obligation Attributes</th>
<th>Obligation 1</th>
<th>Obligation 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Books must be delivered within 5 days of order</td>
<td>Delivered quantities must match ordered quantities</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Temporal</td>
<td>Non-temporal</td>
</tr>
<tr>
<td><strong>Domain variable name</strong></td>
<td>delivery_deadline</td>
<td>delivery_Quantity_Accuracy</td>
</tr>
<tr>
<td><strong>Required performance</strong></td>
<td>5</td>
<td>100 %</td>
</tr>
<tr>
<td><strong>Metric</strong></td>
<td>Temporal – Relative- Day</td>
<td>Number - percentage</td>
</tr>
<tr>
<td><strong>Activity Completion variable</strong></td>
<td>delivery_status</td>
<td>Not Applicable</td>
</tr>
<tr>
<td><strong>Comparator</strong></td>
<td>Less than or equal to</td>
<td>Equal to</td>
</tr>
</tbody>
</table>

Table 5-2: Values of attributes of obligations in term 2 of the library-book vendor contract

We further establish five levels of compliance for both obligations associated with term 2 of the above contract. The threshold values and associated compliance levels for obligation 1 and 2 of term 2 above are presented in tables 5-3 and 5-4 below respectively:
<table>
<thead>
<tr>
<th>Threshold Values</th>
<th>Delivery Status</th>
<th>Resulting Compliance Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 days</td>
<td>True</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-2</td>
</tr>
<tr>
<td>15 days</td>
<td>True</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-3</td>
</tr>
<tr>
<td>25 days</td>
<td>True</td>
<td>-3</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-4</td>
</tr>
<tr>
<td>30 days</td>
<td>True</td>
<td>-4</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>-5</td>
</tr>
</tbody>
</table>

Table 5-3: Compliance levels in obligation 1 of term 2

<table>
<thead>
<tr>
<th>Threshold Values</th>
<th>Resulting Compliance Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>90 % Correct</td>
<td>-1</td>
</tr>
<tr>
<td>80 % Correct</td>
<td>-2</td>
</tr>
<tr>
<td>70 % Correct</td>
<td>-3</td>
</tr>
<tr>
<td>50 % Correct</td>
<td>-4</td>
</tr>
<tr>
<td>&lt;50% Correct</td>
<td>-5</td>
</tr>
</tbody>
</table>

Table 5-4: Compliance levels in obligation 2 of term 2
The figure 5-1 below presents XML representation of library-book vendor ROAD contract that contains the terms discussed above.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Contract endDate="2009-12-31T12:00:00" id="100" name="LibraryBookVendorContract"
startDate="2009-01-01T12:00:00" type="SupplyChain"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <State>Incipient</State>
  <Terms>
    <Term Id="1000" type="request-reponse" validFrom="2009-01-01T12:00:00" validTo="2009-07-01T12:00:00">
      <ObligatedParty>RoleA</ObligatedParty>
      <Description>Library can place order for books from the book-vendor</Description>
      <Obligations>
        <Obligation domainVariableName="maximumBooksPerOrder" id="11"
parentTermReferenceID="1000" status="NA" type="NONTEMPORAL">
          <RequiredPerformance>500</RequiredPerformance>
          <ComplianceFunction>
            <ThresholdValues></ThresholdValues>
          </ComplianceFunction>
          <Metric>
            <ObligationMetric>Number</ObligationMetric>
            <DataType>int</DataType>
          </Metric>
          <Comparator>LessThanEqualTo</Comparator>
          <ActualPerformance>0</ActualPerformance>
          <CurrentComplianceLevel>0</CurrentComplianceLevel>
          <PerformanceProviderReference>www.roadframework.performanceprovider.com</PerformanceProviderReference>
        </Obligation>
        <Obligation domainVariableName="minimumBooksPerOrder" id="12"
parentTermReferenceID="1000" status="NA" type="NONTEMPORAL">
          <RequiredPerformance>5</RequiredPerformance>
          <ComplianceFunction>
            <ThresholdValues></ThresholdValues>
          </ComplianceFunction>
        </Obligation>
      </Obligations>
    </Term>
  </Terms>
</tns:Contract>
```

Figure 5-1: XML Representation of library-book vendor contract
Figure 5-1: XML Representation of library-book vendor contract (Cont)
<Obligation domainVariableName="deliveryQuantity" id="102" parentTermReferenceID="1001" status="NA" type="NONTEMPORAL">
  <RequiredPerformance>100</RequiredPerformance>
  <ComplianceFunction>
    <ThresholdValues>
      <ThresholdValue>90</ThresholdValue>
      <ThresholdValue>80</ThresholdValue>
      <ThresholdValue>70</ThresholdValue>
      <ThresholdValue>50</ThresholdValue>
    </ThresholdValues>
    <Metric>
      <ObligationMetric>Percentage</ObligationMetric>
      <DataType>double</DataType>
    </Metric>
    <Comparator>GreaterThanEqualTo</Comparator>
    <ActualPerformance>0</ActualPerformance>
    <CurrentComplianceLevel>0</CurrentComplianceLevel>
    <PerformanceProviderReference>www.roadframework.performanceprovider.com</PerformanceProviderReference>
    <ObligationECARules>
      ComplianceRules.drls
    </ObligationECARules>
  </ComplianceFunction>
</Obligation>
</Obligations>
</TermECARules>
</Term>
</Terms>
</GeneralClauses>
</tns:Contract>

Figure 5-1: XML Representation of library-book vendor contract (Cont)
5.3 Implementation of ROAD contract with Java

Our test application as discussed above is built in java. We use JAXB (Java Architecture for XML Binding) [28] for generating java classes from XML schema of ROAD contract. We then populate the generated classes with domain specific data by unmarshalling the ROAD Contract XML file presented in figure 5-1. The following figure 5-2 presents the overview of the ROAD contract schema generated classes. The implementation of various methods required by these classes, for example for calculating compliance levels is built into separate set of classes that mirror the schema generated classes. At runtime we create instance our implementation of Contract classes with data from schema generated classes.

![Figure 5-2: Overview of java classes generated from xml schema of ROAD contract](image)

5.4 Using Drools engine to define and execute rules

As mentioned earlier we used drools rule engine to define and execute the ECA rules for obligations, terms and general clauses of library-book vendor contract. Drools [27] is an
enhanced rule engine implementation based on Rete algorithm [29] and tailored for java programming language. In drools, business rules are defined in a rules file that uses .drl or .drls extension. This file is read by drools engine at runtime and rules are triggered when conditions evaluations returns true. Each rule contains a part called LHS (or conditions) and RHS (or actions). A rule is triggered when LHS or conditions are satisfied and the result is the execution of actions specified in the RHS portion. The rules in drools are written in language native to drools and it uses non XML format. Drools also provides support for defining domain specific languages for writing rules. We discuss the use of domain specific language later as we define rules for library-book vendor contract.

5.4.1 ECA rules for general clauses of library-book vendor contract

General clauses enforce business rules by executing some actions in response to the compliance results of obligations of terms of the contract. The table 5-5 below presents some of the business rules identified in general Clauses of library-book vendor contract.

| Rule 1: If an order contains books quantities less than 5 or greater than 500. |
|-----------------------------|---------------------------------|
| Event:                      | Compliance notification         |
| Condition:                  | If (term1.obligation1.complianceLevel = 0 OR term1.obligation2.complianceLevel = 0) |
| Action:                     | Send message to library informing about invalid order. |

| Rule 2: If books are not delivered by the book vendor after 5 days of order. |
|-----------------------------|---------------------------------|
| Event:                      | Compliance notification         |
| Condition:                  | If (term2.obligation1.complianceLevel = -1 AND term2.obligation1.status = “CHECKED”) |
| Action:                     | Send message to book vendor about missed delivery deadline. No effect on the state of the contract. |

| Rule 3: If books are not delivered by the book vendor after 15 days of order. |
|-----------------------------|---------------------------------|
| Event:                      | Compliance notification         |
| Condition:                  | If (term2.obligation1.complianceLevel = -3 AND |
term2.obligation1.status = "CHECKED")

**Action:**
Send message to book vendor and composite organizer. Change contract state to Active (Major-breach), charge $100 as penalty to the book vendor for late delivery.

**Rule 4:** *If books are not delivered by the book vendor after 20 days of order.*

<table>
<thead>
<tr>
<th>Event:</th>
<th>Compliance notification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Condition:</strong></td>
<td>If (term2.obligation1.complianceLevel = -4 AND term2.obligation1.status = “CHECKED”)</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Send messages to composite organizer and highest management contact in book vendor. Change contract state to Suspended and penalty still applies.</td>
</tr>
</tbody>
</table>

**Rule 5:** *If books are not delivered by the book after 30 days of order.*

<table>
<thead>
<tr>
<th>Event:</th>
<th>Compliance notification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Condition:</strong></td>
<td>If (term2.obligation1.complianceLevel = -5 AND term2.obligation1.status = “CHECKED”)</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Send messages to composite organizer and appropriate department in book vendor. Change contract state to Terminated, penalty still applies but contract now must be re-negotiated before being re-activated.</td>
</tr>
</tbody>
</table>

| Table 5-5: Business rules express as ECA rules in general clauses of above contract |

In the above rules though we have specified application of $100.00 penalty as one of the actions taken by general clauses, the current design does not support enforcement of such penalties. For now we assume it is handled by another layer and we leave it for future work. Also notice that we are checking the status of obligation 1 of term 2 inside conditions in the above table. This is because we need to check the status of temporal obligations in order to verify if the underlying activity has been completed. As discussed in chapter 3 under section 3.3.2.8 the temporal obligations have status value “CHECKED” when underlying activity is incomplete and this value changes to “TERMINATED” upon completion of activity. The underlying activity in term 2 above is the delivery of books.
In order to use drools rule engine to define and execute above business rules, we define a domain specific language for expressing the above business rules in a rules file called ComplianceRules.drls. This is not a requirement and we also expressed the above rules without using domain specific language. Please refer to section 1 in the appendix at the end of the thesis that presents the above rules in native rules language that mostly uses Java programming language syntax. We defined Domain Specific Language (DSL) for our test application as it offers several advantages. DSL acts as a template for conditions and actions part of the rules where the same template can be used several times with different parameters. It also enables authoring and reading of rules by non technical people who do not understand java. Further it separates definition of rules from implementation of rules. The table 5-6 below illustrates mapping between some of the DSL expressions defined for library-book vendor application and the corresponding drools rule language expression.

<table>
<thead>
<tr>
<th>DSL Language Expression</th>
<th>Rule Language Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is an Event with</td>
<td>Event()</td>
</tr>
<tr>
<td>- eventType is '{type}'</td>
<td>e.eventType == '{type}'</td>
</tr>
<tr>
<td>- eventSource is '{source}'</td>
<td>e.eventSource == '{source}'</td>
</tr>
<tr>
<td>- termID is {ID}</td>
<td>e.termID == {ID}</td>
</tr>
<tr>
<td>- obligationID is {obID} compliance level is {complianceLevel} status is '{status}'</td>
<td>(eval(e.getComplianceObject({obID}, {complianceLevel}, &quot;{status}&quot;), != null))</td>
</tr>
</tbody>
</table>

Table 5-6: Mapping between DSL language expressions and drools native rule language

The event in the above table is a java object that contains the following attributes: eventType, eventSource, termID and a list of compliance objects. Each compliance object further contains an obligationID, compliance level, status and type of an obligation. At runtime we insert this object into the working memory of rules engine and rules are triggered when values in LHS of rules match the values of attributes of the event object.

We define the rules for the drools rule engine below using the DSL expressions in table 5-6 above.
Rule 1: If an order contains books quantities less than 5 or greater than 500.

```plaintext
rule "General Clauses term 1000 violation of obligation 11 or 12"
when
    There is an Event with
    - eventType is 'NOTIFICATION'
    - eventSource is 'TERM'
    - termID is 1000
    - TWO OBLIGATIONS OR TOGETHER
        obligationID is 11 compliance level is 0 status is 'TERMINATED'
        obligationID is 12 compliance level is 0 status is 'TERMINATED'
then
    sendMessageToRoleA("Invalid order, check contract ordering term and place order again");
end
```

Figure 5-3: General Clauses Rule 1 definition

Rule 2: If books are not delivered by the book vendor after 5 days of order.

```plaintext
rule "General Clauses term 1001 obligation 101 incomplete with compliance level -1"
when
    There is an Event with
    - eventType is 'NOTIFICATION'
    - eventSource is 'TERM'
    - termID is 1001
    - obligationID is 101 compliance level is -1 status is 'CHECKED'
then
    sendMessageToRoleA("Please deliver ASAP");
end
```

Figure 5-4: General Clauses Rule 2 definition

Rule 3: If books are not delivered by the book vendor after 15 days of order.

```plaintext
rule "General Clauses term 1001 obligation 101 incomplete with compliance level -3"
when
    There is an Event with
    - eventType is 'NOTIFICATION'
    - eventSource is 'TERM'
    - termID is 1001
    - obligationID is 101 compliance level is -3 status is 'CHECKED'
then
    updateContractState(myROADContract, "ActiveMajorBreach");
    sendMessageToRoleB("Delivery books ASAP with penalty of $100");
    sendMessageToCompositeOrganizer("Term 1001, obligation 101 is at compliance level -3, $100 penalty applies");
end
```

Figure 5-5: General Clauses Rule 3 definition
Rule 4: *If books are not delivered by the book vendor after 20 days of order.*

<table>
<thead>
<tr>
<th>rule</th>
<th>&quot;General Clauses term 1001 obligation 101 incomplete with compliance level -4&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
<td>There is an Event with</td>
</tr>
<tr>
<td></td>
<td>- eventType is 'NOTIFICATION'</td>
</tr>
<tr>
<td></td>
<td>- eventSource is 'TERM'</td>
</tr>
<tr>
<td></td>
<td>- termID is 1001</td>
</tr>
<tr>
<td></td>
<td>- obligationID is 101 compliance level is -4 status is 'CHECKED'</td>
</tr>
<tr>
<td>then</td>
<td>updateContractState(myROADContract, &quot;Suspended&quot;);</td>
</tr>
<tr>
<td></td>
<td>sendMessageToRoleB(&quot;Contract is suspended unless delivery is made, Delivery books ASAP with penalty of $100&quot;);</td>
</tr>
<tr>
<td></td>
<td>sendMessageToCompositeOrganizer(&quot;Term 1001, obligation 101 is at compliance level -3, $100 penalty applies&quot;);</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-6: General Clauses Rule 4 definition

Rule 5: *If books are not delivered by the book after 30 days of order.*

<table>
<thead>
<tr>
<th>rule</th>
<th>&quot;General Clauses term 1001 obligation 101 incomplete with compliance level -5&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
<td>There is an Event with</td>
</tr>
<tr>
<td></td>
<td>- eventType is 'NOTIFICATION'</td>
</tr>
<tr>
<td></td>
<td>- eventSource is 'TERM'</td>
</tr>
<tr>
<td></td>
<td>- termID is 1001</td>
</tr>
<tr>
<td></td>
<td>- obligationID is 101 compliance level is -5 status is 'CHECKED'</td>
</tr>
<tr>
<td>then</td>
<td>updateContractState(myROADContract, &quot;TERMINATED&quot;);</td>
</tr>
<tr>
<td></td>
<td>sendMessageToRoleB(&quot;Contract terminated due to severe non compliance with term 2 of the contract&quot;);</td>
</tr>
<tr>
<td></td>
<td>sendMessageToCompositeOrganizer(&quot;Term 1001, obligation 101 is at compliance level -5 and status terminated, contract has been terminated&quot;);</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-7: General Clauses Rule 5 definition

In rules 1 to 5 defined above for general clauses we are checking for event type, event source, term ID, obligation ID, compliance level and status of an obligation in the “when” part of the rule by using the DSL expressions defined in table 5-6. The “then” part specifies the action taken when the “when” part evaluates to true. In the above rules we invoke java methods in the “then” part to update contract state and send messages to composite organizer and contracted parties. We can also specify DSL expressions for “then” part of the rule similar to “when” part and use in the rule definitions in the same way.
5.4.2 ECA rules for terms of library-book vendor contract

For this application, we defined two generic rules that are invoked for all the terms. Rules specific to a particular term can be defined by adding another constraint to the rules below with a term ID. For this application we require each term to behave in similar way so we did not add term ID constraint. The two rules used by the terms for our application are as follows:

Rule 1: This rule is triggered by general clauses in order to force compliance evaluation of obligations of a term.

```
rule "Term Evaluation Invocation"
  no-loop true
  when
    There is an Event with
    - eventType is 'EVALUATION'
    - eventSource is 'GENERALCLAUSES'
  then
    populateComplianceObjects(myROADContract, e);
    modify (e) {
      setEventSource("TERM"),
      setEventType("EVALUATION")
    }
  end
```

Figure 5-8: Terms Rule 1 definition

Rule 2: This rule is triggered by obligations of a term in order to escalate their compliance evaluation results.

```
rule "Term Notification Invocation"
  no-loop true
  when
    There is an Event with
    - eventType is 'NOTIFICATION'
    - eventSource is 'OBLIGATION'
  then
    evaluateReportedCompliance(myROADContract, e);
    modify (e) {
      setEventSource("TERM"),
      setEventType("NOTIFICATION")
    }
  end
```

Figure 5-9: Terms Rule 2 definition
5.4.3 ECA rules for obligations of library-book vendor contract

For obligations we need only two rules again. As discussed in chapter 4, on being triggered obligations evaluate and escalate the compliance to terms without any other condition evaluations. The obligations are triggered either with a timer or from a term but action performed in both cases is the same as shown below.

Rule 1: Invocations from term

<table>
<thead>
<tr>
<th>Rule</th>
<th>&quot;Obligation Invocation from term&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
<td>There is an Event with</td>
</tr>
<tr>
<td></td>
<td>- eventType is 'EVALUATION'</td>
</tr>
<tr>
<td></td>
<td>- eventSource is 'TERM'</td>
</tr>
</tbody>
</table>
| then | evaluateCompliance(myROADContract, e);
|      | modify (e) {                   |
|      |   setEventSource("OBLIGATION"), |
|      |   setEventType("NOTIFICATION") |
| end  |                                 |

Rule 2: Invocations from a timer

<table>
<thead>
<tr>
<th>Rule</th>
<th>&quot;Obligation Invocation from timer&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>when</td>
<td>There is an Event with</td>
</tr>
<tr>
<td></td>
<td>- eventType is 'EVALUATION'</td>
</tr>
<tr>
<td></td>
<td>- eventSource is 'TIMER'</td>
</tr>
</tbody>
</table>
| then | evaluateCompliance(myROADContract, e);
|      | modify (e) {                   |
|      |   setEventSource("OBLIGATION"), |
|      |   setEventType("NOTIFICATION") |
| end  |                                 |

5.5 Running the test application

In this section we demonstrate run time behavior of our test application that verifies the contract design proposed in this thesis. We start our application by reading ROAD contract XML schema that is used by JAXB API for generating java classes. The ROAD contract XML file (Contract.xml) is then unmarshalled in order to populate the java objects at runtime. The code in figure 5-12 illustrates these points.

```
JAXBContext jc = JAXBContext.newInstance("ROAD.ComplianceContracts.xml.bindings");
Unmarshaller unmarshaller = jc.createUnmarshaller();
ContractType contractBinding = (ContractType) ((JAXBElement) unmarshaller.unmarshal(new File("data/Contract.xml"))).getValue();
```

Figure 5-10: Obligation Rule 1 definition

Figure 5-11: Obligation Rule 2 definition

Figure 5-12: Generating Java classes using JAXB
After creating instance of contractBinding object shown in the code in figure 5-12, we create an instance of ROAD contract (contractImpl) whose data structure mirrors contractBinding object but it also contains all the required method implementations.

The code in figure 5-13 below defines a static method called readRule() that builds the rule base by reading the rules file i.e. ComplianceRules.dslr and DSL expressions file i.e. ComplianceRules.dsl. As discussed earlier the rules file contains all our rules for general clauses, terms and obligations. The dsl file contains the mapping information between the domain specific language expressions used in the rules file and native language of drools engine.

```java
private static RuleBase readRule() throws Exception {
    Reader source = new InputStreamReader(TestApplication.class.getResourceAsStream("Rules/ComplianceRules.dslr"));
    Reader dsl = new InputStreamReader(TestApplication.class.getResourceAsStream("Rules/ComplianceRules.dsl"));
    PackageBuilder builder = new PackageBuilder();
    builder.addPackageFromDrl(source, dsl);
    Package pkg = builder.getPackage();
    RuleBase ruleBase = RuleBaseFactory.newRuleBase();
    ruleBase.addPackage(pkg);
    return ruleBase;
}
```

Figure 5-13: Using Drools API to build rule base

The code in lines 1, 2, and 9 in the following figure 5-14 creates an instance of ROAD contract named ActiveContract, sets its state to “ACTIVE” and inserts it into working memory as a global variable. We get an instance of working memory of rule base in lines 3 and 4. In lines 4, 6, 7 and 8 we create an event object instance and give value of “GENERALCLAUSES” to eventSource attribute, “EVALUATION” to eventType attribute and 1000 to termID in order to invoke term with ID 1000 at runtime. In line 10 we insert event object into the working memory of rules base and line 11 fires all rules in the working memory on the event object.
In the above code we start our application by triggering term with ID 1000 that states: *Library can place order for books to the book vendor.* The event object inserted into the working memory in figure 5-14 above has term ID set to 1000 and it triggers the rule 1 of terms presented in figure 5-8. This rule identifies various obligations associated with this term and triggers the rule 1 of obligations presented in figure 5-10. The triggering of obligation rule invokes compliance evaluation of two obligations of the term. The two obligations associated with this term verify that the delivery quantity is within the allowed range of 5 and 500. The first obligation checks if the delivery quantity is more than 5 and second if it is less than 500. As we do not have implementation of performance providers at the current stage we are specifying the actual performance values of obligations in the XML file and at runtime compliance level is evaluated by reading this value. The results of compliance evaluation are stored in the attributes “complianceLevel”, “obligationID”, “status” of the compliance object inside the event object. After compliance evaluation is complete this rule invokes the rule 2 of terms presented in figure 5-9 by changing the values of eventSource and eventType attributes of the event object to “OBLIGATION” and “NOTIFICATION” respectively. As there are no further checks specified in the rule 2 of terms, this rule simply triggers general clauses rules of term 1000 by changing values of eventSource and eventType to “TERM” and “NOTIFICATION” respectively. These changes on event object can now trigger the general clauses rule presented in figure 5-3 if the compliance levels of one of the obligations of this term was “0”, otherwise the rule in figure 5-15 below is triggered. We
set the actual performance values of obligations of this term to 100. The value of 100 satisfies the required performance values of both obligations resulting in the triggering of rule in figure 5-15 below. The java code that implements the evaluation of compliance levels of obligations is included in section 3 of the appendix at the end of the thesis. The screen dump in figure 5-16 presents invocations of various terms, obligations and general clauses discussed so far. We simply added a print line statement to each rule to illustrate when it is being triggered. Please note that line 4 in figure 5-16 is printed due to triggering of rule in figure 5-15.

```java
//this rule is triggered when quantity of books in the order complies with contract requirements
rule "General Clauses term 1000 full compliance with obligation 11 and 12"
    when
        There is an Event with
            - eventType is 'NOTIFICATION'
            - eventSource is 'TERM'
            - termID is 1000
            - TWO OBLIGATIONS AND TOGETHER
                obligationID is 11 compliance level is 1 status is 'TERMINATED'
                obligationID is 12 compliance level is 1 status is 'TERMINATED'
    then
        System.out.println("Full compliance by Library - Ordered quantities are within the allowed range");
        modify (e) {
            setEventSource("GENERALCLAUSES"),
            setEventType("EVALUATION"),
            setTermID(1001)
        }
end
```

Figure 5-15: General Clauses rule for full compliance with term 1000

1. Rule 1 of terms invoked, term ID is 1000
2. OBLIGATION rule invoked
3. Rule 2 of terms invoked, term ID is 1000

4. Full compliance by Library - Ordered quantities are within the allowed range

Figure 5-16: Screen dump displaying triggering of rules for terms, obligations and general clauses for term 1000

Also note that in figure 5-15 that we are setting term ID to 1001 in “then” part of the rule in order to trigger evaluation of term 1001 after term 1000 finishes execution. As
discussed previously, conversation clauses control the execution of terms by checking the order of execution of terms. As we don’t have complete implementation of conversation clauses at the current stage so we trigger execution of next term in our test application from the current term under execution in order to demonstrate implementation of our design.

In order to demonstrate different compliance level evaluations by obligations and triggering of different rules in general clauses in response to different compliance levels, we will now invoke rules on term 1001 with different actual performance values for obligation 101. The obligation 101 specifies delivery deadline for term 1001. The required performance value of this obligation is 5 and this obligation has a compliance function that holds four threshold values of 10, 15, 20 and 30. The figure 5-17 on next page shows execution of term 1001 with different actual performance values for obligation 101. The obligation returns different compliance levels for different actual performance values and that results in triggering of different rules in general clauses.

In figure 5-17 below all lines in italics illustrate triggering of rules in general clauses. The lines 1 to 4 illustrate invocation of rules for terms, obligations and general clauses for term 1000 as discussed before. From lines 5 to 43 we trigger evaluation of term 1001 with different actual performance values for obligation 101. Obligation 101 is a temporal obligation so it is evaluated until the underlying activity, i.e. delivery of books is made or final compliance level is returned by the obligation.

At line 5 we trigger term 1001 with actual performance value of 4 and we set the status of underlying activity (delivery_status) to false. Triggering obligation evaluation with these values returns compliance level 1 as delivery deadline of 5 days hasn’t arrived and therefore obligation hasn’t been violated.

At line 9 we again trigger this term with actual performance value of 10 and with delivery_status set to false. Now general clauses receive compliance level -1 and but no action is taken by general clauses as it is a very low level breach.
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Rule 1 of terms invoked, term ID is 1000</td>
</tr>
<tr>
<td>2.</td>
<td>OBLIGATION rule invoked</td>
</tr>
<tr>
<td>3.</td>
<td>Rule 2 of terms invoked, term ID is 1000</td>
</tr>
<tr>
<td>4.</td>
<td>Full compliance by Library - Ordered quantities are within the allowed range</td>
</tr>
<tr>
<td>5.</td>
<td>Rule 1 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>6.</td>
<td>OBLIGATION rule invoked</td>
</tr>
<tr>
<td>7.</td>
<td>Rule 2 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>8.</td>
<td>Delivery pending but deadline hasn’t passed yet</td>
</tr>
<tr>
<td>9.</td>
<td>Rule 1 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>10.</td>
<td>OBLIGATION rule invoked</td>
</tr>
<tr>
<td>11.</td>
<td>Rule 2 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>12.</td>
<td>Current Contract state is: ACTIVE</td>
</tr>
<tr>
<td>13.</td>
<td>Delivery still pending with current compliance level -1</td>
</tr>
<tr>
<td>14.</td>
<td>Rule 1 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>15.</td>
<td>OBLIGATION rule invoked</td>
</tr>
<tr>
<td>16.</td>
<td>Rule 2 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>17.</td>
<td>Contract state was ACTIVE</td>
</tr>
<tr>
<td>18.</td>
<td>New Contract state is ActiveMinorBreach</td>
</tr>
<tr>
<td>19.</td>
<td>Message to books vendor - Delivery books ASAP with penalty of $100</td>
</tr>
<tr>
<td>20.</td>
<td>Composite organizer updated - Term 1001, obligation 101 is at compliance level -2</td>
</tr>
<tr>
<td>21.</td>
<td>Delivery still pending with current compliance level -2</td>
</tr>
<tr>
<td>22.</td>
<td>Rule 1 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>23.</td>
<td>OBLIGATION rule invoked</td>
</tr>
<tr>
<td>24.</td>
<td>Rule 2 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>25.</td>
<td>Contract state was ActiveMinorBreach</td>
</tr>
<tr>
<td>26.</td>
<td>New Contract state is ActiveMajorBreach</td>
</tr>
<tr>
<td>27.</td>
<td>Message to books vendor - Delivery books ASAP with penalty of $100</td>
</tr>
<tr>
<td>28.</td>
<td>Composite organizer updated - Term 1001, obligation 101 is at compliance level -3</td>
</tr>
<tr>
<td>29.</td>
<td>Delivery still pending with current compliance level -3</td>
</tr>
<tr>
<td>30.</td>
<td>Rule 1 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>31.</td>
<td>OBLIGATION rule invoked</td>
</tr>
<tr>
<td>32.</td>
<td>Rule 2 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>33.</td>
<td>Contract state was ActiveMajorBreach</td>
</tr>
<tr>
<td>34.</td>
<td>New Contract state is SUSPENDED</td>
</tr>
<tr>
<td>35.</td>
<td>Message to books vendor - Delivery books ASAP with penalty of $100</td>
</tr>
<tr>
<td>36.</td>
<td>Composite organizer updated - Term 1001, obligation 101 is at compliance level -4</td>
</tr>
<tr>
<td>37.</td>
<td>Delivery still pending with current compliance level -4</td>
</tr>
<tr>
<td>38.</td>
<td>Rule 1 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>39.</td>
<td>OBLIGATION rule invoked</td>
</tr>
<tr>
<td>40.</td>
<td>Rule 2 of terms invoked, term ID is 1001</td>
</tr>
<tr>
<td>41.</td>
<td>Delivery made with Compliance level -4, - FINAL COMPLIANCE</td>
</tr>
<tr>
<td>42.</td>
<td>Contract state was SUSPENDED</td>
</tr>
<tr>
<td>43.</td>
<td>New Contract state is ACTIVE</td>
</tr>
</tbody>
</table>

Figure 5-17: Screen dump displaying triggering of rules for terms, obligations and general clauses for term 1000 and 1001
At line 14 we trigger compliance evaluation with actual performance value of 15 and delivery status false. This time general clauses get compliance level -2. General clauses change contract state to “ActiveMinorBreach” and send messages to book vendor and ROAD composite organizer in lines 17 to 21.

Similarly compliance evaluation is triggered in lines 22 and 30 with different actual performance values and general clauses take different actions as shown in italics for each compliance result changing contract state to “ActiveMajorBreach” in line 26 and to “SUSPENDED” in line 34.

In line 38 compliance evaluation is triggered with actual performance value of 30 but with delivery status set to true. This time general clauses receive final compliance level of the temporal obligation. General clauses now re-activate the contract by bringing it back from “SUSPENDED” state to “ACTIVE” as underlying activity i.e. delivery of books has been completed by the book vendor. General clauses also establish final compliance level of -4 for obligation 101. As discussed in the previous chapter the actions taken by general clauses in response to different compliance levels vary depending on the requirements of individual applications. We have included definition of each of the rules triggered in figure 5-17 in section 2 of the appendix at the end of the thesis.

5.6 Summary

In this chapter we presented the design and implementation details of our test application. We described the development of prototype based on the compliance model proposed in this thesis. In the end we demonstrated the behaviour of our prototype at runtime in accordance with the proposed compliance model.
6. Conclusion

This chapter concludes the thesis by discussing its contribution to ROAD framework and to the field of software engineering. We also discuss some limitations of current design and future work that needs to be done to address those limitations.

6.1 Contribution

The major contribution of this thesis is the development of design and implementation of life cycle of ROAD contracts based on the level of fulfillment of performance requirements of its terms.

ROAD framework like other contract based adaptive software system frameworks [14, 15 and 16] provided only a preliminary design for its contracts without sufficient details. Also none of the electronic contract frameworks discussed in chapter 2 describes monitoring of contract performance with its effect on the external state of the contract as most of the approaches rely on some third party services for monitoring runtime execution of contracts and neither approach describes how these third party services are actually implemented. This thesis not only illustrates how to monitor execution of terms of ROAD contracts at runtime but also provides a mechanism for handling breaches to contract terms according to the precise application specific requirements. We contribute to the field of software engineering as we describe in detail the relationship between the external state of the contract and level of fulfillment of performance requirements of terms of the contract and this can also be applied to other electronic and software contract systems. The compliance model proposed in this thesis can also be applied to other software and electronic contract systems.

This thesis extends the layered model of contracts proposed in ROAD framework by providing formal definitions of general clauses, terms and obligations and describing relationships between these layers. We developed formal definitions of general clauses, terms and obligations using XML. We modeled terms and obligations based on the
concepts of deontic logic and temporal constraints and described how ROAD contracts express and enforce permissions, obligations and prohibitions of contracted parties. We established relationships between various layers of ROAD contract by associating ECA rules with each layer where triggering of events and execution of subsequent actions in one layer leads to chained execution of more events and actions in other layers.

We introduced compliance model that provides flexibility to ROAD contracts while handling violations to contract terms. The compliance model enables handling contract breaches according to precise business requirements. The proposed design provides a mechanism that enables execution of different actions according to the degree of deviation of actual performance from required performance as contract designer can specify selective actions inside ECA rules of general clauses according to the level of compliance of obligations. The proposed compliance model further enables contract designers to define life cycle of ROAD contracts according to the requirements of individual applications. We also described the collaboration between ROAD composite organizer and general clauses of contracts and proposed that the level of control granted to general clauses over the life cycle of ROAD contract depends on the desired level of automation in an application.

6.2 Limitations and Future work

Expressing obligations on conversation clauses
The obligations proposed in this thesis express and measure performance requirements of terms. This thesis did not address conversation clauses that are also very much a part of ROAD contracts as some work addressing these clauses has already been done by Pham et. al. in [2]. But further work is required for expressing obligations proposed in this thesis against conversation clauses. When obligations are expressed against terms as well as conversation clauses further thought must be given to the types of obligations that should be associated each of these layers.
Constraining contract terms with external events

We emphasize in the thesis that terms of contracts by themselves only express permissions granted to contracted parties. In order to express and enforce obligations of contracted parties, terms of contracts must be associated with a temporal obligation. A term can also express an obligation if bound by an external event other than a temporal obligation. For example in the thesis we discussed the following term several times: *Books to be delivered by the book vendor to the library*, and temporal obligation associated with this term stated that: *books must be delivered within 5 days of order*. Another type of constraint on this term can be that: *books must be delivered by the book vendor within 5 days of being released by the publisher*. But we leave this aspect of obligation to future work as this type of obligation enforcement requires monitoring of external events and we are only modeling events represented by messages passing through a ROAD composite in this study.

Complex type metric attribute of obligations

This study limits expression of metrics against obligations to simple types or the three types of temporal metrics discussed in section 3.3.1.1. We do not address complex type metrics that allow performance measurement by comparing complex types in this thesis. This is a desired feature for real world business applications that may require comparison between some domain specific objects and therefore needs to be explored further.

Enforcing penalties on parties responsible for contract breaches

In this thesis though we specified application of penalties to parties when they breach their contractual requirements as an action that can be taken by general clauses after evaluating compliance results of an obligation, we did not cover the enforcement of penalties in this study and have left it for future work.

Prototype implementation

The prototype described in chapter 5 is developed as a proof of concept. Some aspects of the design proposed in the thesis have not been implemented. The design of obligations proposed in chapter 3 allows contract designers to specify the underlying data type of
required performance and actual performance attributes declaratively. The prototype implementation only supports string and double data types. The ECA rules associated with general clauses, terms and obligations are expressed using native language of drools engine and a domain specific language defined for the test application inside a separate rules file. The rules file is separate from the XML representation of contract. Also the prototype requires the person defining the rules to have some understanding of Java programming language. The proposed design requires invocation of individual terms through conversation clauses but as we do not have complete implementation of conversation clauses, at current stage we manually trigger each term or trigger one term from another in the test application.

Hence, before ROAD contracts can be truly used to handle real world business relationships we need to express obligations proposed in this thesis against conversation clauses and clearly define the role played by conversation clauses in the compliance model. Further research is also required to enable ROAD contracts to capture and respond to events that are external to the ROAD composite. ROAD contracts also require complex metric types for performance requirements of its terms and ability to express such metrics declaratively. Future work also needs to establish how ROAD contracts can enforce penalties on parties that breach their contractual requirements.
Appendix

1. Rules defined in native drools language

The following two rules are for general clauses of term with ID 1000 of library-book vendor contract and are defined using native rules language of drools engine. Please note that the syntax used in defining these rules is similar to Java programming language.

The following rule is triggered when quantity of books in the order complies with contract requirements

```java
rule "General Clauses term 1000 full compliance with obligation 11 and 12"
no-loop true

when
e : Event(eventType == "NOTIFICATION" && eventSource == "TERMS" && termID == "1000"
&& (eval(e.getComplianceObject(11) != null)) &&
(eval(e.getComplianceObject(11).getStatus() == "TERMINATED"))
&& (eval(e.getComplianceObject(12) != null)) &&
(eval(e.getComplianceObject(12).getStatus() == "TERMINATED"))
&& (eval(e.getComplianceObject(11).getComplianceLevel() == 1) &&
(eval(e.getComplianceObject(12).getComplianceLevel() == 1)))

then
System.out.println("Full compliance by Library - Ordered quantities are within the allowed range");
modify (e) {
    setEventSource("GENERALCLAUSES"),
    setEventType("EVALUATION"),
    setTermID(1001)
}

end
```
The following rule is triggered when quantity of books in the order breach contract requirements

**rule** "General Clauses term 1000 violation of obligation 11 or 12"
**no-loop** true

**when**

\[
\begin{align*}
&\text{e : Event(eventType == "NOTIFICATION" && eventSource == "TERMS" && termID == "1000"}
&& (\text{eval(e.getComplianceObject(11) != null)) &&}
&& (\text{eval(e.getComplianceObject(11).getStatus() == "TERMINATED")}
&& (\text{eval(e.getComplianceObject(12) != null)) &&}
&& (\text{eval(e.getComplianceObject(12).getStatus() == "TERMINATED")}
&& ((\text{eval(e.getComplianceObject(12).getComplianceLevel() == 0)}) ||
((\text{eval(e.getComplianceObject(12).getComplianceLevel() == 0)})
\end{align*}
\]

**then**

System.out.println("Invalid order as ordered quantities of books fall outside the allowed range, please check contract terms and conditions");

**end

2. Definition of rules triggered in figure 5-17 in chapter 5

The following six general clauses rules are triggered in the screen dump of our test application as shown in figure 5-17.

2.1 General Clauses rule triggered at line 8 in figure 5-17.

**rule** "General Clauses term 1001 obligation 101 compliance evaluation before delivery deadline results in incomplete obligation"
**no-loop** true

**when**

\[
\begin{align*}
e: \text{Event()}
\text{There is an Event with}
- \text{eventType is 'NOTIFICATION'}
- \text{eventSource is 'TERM'}
- \text{termID is 1001}
- \text{obligationID is 101 compliance level is 1 status is 'CHECKED'
\end{align*}
\]

**then**
2.2 General Clauses rule triggered at line 12 in figure 5-17.

rule "General Clauses term 1001 obligation 101 incomplete with compliance level -1"
no-loop true
    when
e: Event()
    There is an Event with
    - eventType is 'NOTIFICATION'
    - eventSource is 'TERM'
    - termID is 1001
    - obligationID is 101 compliance level is -1 status is 'CHECKED'
    then
      System.out.println("Current Contract state is: " + myROADContract.getGeneralClauses().getState());
      System.out.println("Delivery still pending with current compliance level -1, - INCOMPLETE OBLIGATION");
end

2.3 General Clauses rule triggered at line 17 in figure 5-17.

rule "General Clauses term 1001 obligation 101 incomplete with compliance level -2"
no-loop true
    when
e: Event()
    There is an Event with
    - eventType is 'NOTIFICATION'
    - eventSource is 'TERM'
    - termID is 1001
    - obligationID is 101 compliance level is -2 status is 'CHECKED'
    then
      updateContractState(myROADContract, "ActiveMinorBreach");
      sendMessageToRoleB("Message to books vendor - Delivery books ASAP with penalty of $100");
      sendMessageToCompositeOrganizer("Composite organizer updated - Term 1001, obligation 101 is at compliance level -2");
      System.out.println("Delivery still pending with current compliance level -2, - INCOMPLETE OBLIGATION");
End
2.4 General Clauses rule triggered at line 25 in figure 5-17.

```
rule "General Clauses term 1001 obligation 101 incomplete with compliance level -3"
no-loop true
when
e: Event()
   There is an Event with
   - eventType is 'NOTIFICATION'
   - eventSource is 'TERM'
   - termID is 1001
   - obligationID is 101 compliance level is -3 status is 'CHECKED'
then
   updateContractState(myROADContract, "ActiveMajorBreach");
   sendMessageToRoleB("Message to books vendor - Delivery books ASAP with penalty of $100");
   sendMessageToCompositeOrganizer("Composite organizer updated - Term 1001, obligation 101 is at compliance level -3");
   System.out.println("Delivery still pending with current compliance level -3, - INCOMPLETE OBLIGATION");
end
```

2.5 General Clauses rule triggered at line 33 in figure 5-17.

```
rule "General Clauses term 1001 obligation 101 incomplete with compliance level -4"
no-loop true
when
e: Event()
   There is an Event with
   - eventType is 'NOTIFICATION'
   - eventSource is 'TERM'
   - termID is 1001
   - obligationID is 101 compliance level is -4 status is 'CHECKED'
then
   updateContractState(myROADContract, "SUSPENDED");
   sendMessageToRoleB("Message to books vendor - Delivery books ASAP with penalty of $100");
   sendMessageToCompositeOrganizer("Composite organizer updated - Term 1001, obligation 101 is at compliance level -4");
   System.out.println("Delivery still pending with current compliance level -4, - INCOMPLETE OBLIGATION");
```
2.6 General Clauses rule triggered at line 41 in figure 5-17.

```java
rule "General Clauses term 1001 obligation 101 terminated with compliance level -4"
no-loop true
  when
    e: Event()
    There is an Event with
    - eventType is 'NOTIFICATION'
    - eventSource is 'TERM'
    - termID is 1001
    - obligationID is 101 compliance level is -4 status is 'TERMINATED'
  then
    System.out.println("Delivery made with Compliance level -4, - FINAL COMPLIANCE");
    updateContractState(myROADContract, "ACTIVE");
end
```

3. Java code that implements compliance evaluation in obligations

3.1 Evaluating non-temporal compliance

```java
public int evaluateNonTemporalCompliance()
{
    // checking if the obligation has an associated compliance function
    boolean containsMultipleComplianceLevels = false;
    if(complianceFunction != null)
    {
        containsMultipleComplianceLevels = true;
    }
    String dataType = metric.getDataType();
    // applying comparison according to the comparator in the XML file
    if(comparator.equalsIgnoreCase("lessthanequalto"))
    {
        if(Double.parseDouble(actualPerformance) <= Double.parseDouble(requiredPerformance))
        {
            currentCompliance = 1;
        }
        else if (containsMultipleComplianceLevels == true)
        { // calling compliance function to get compliance level
            currentCompliance = complianceFunction.getComplianceLevel(actualPerformance);
        }
    }
    else if (containsMultipleComplianceLevels == true)
    {
```

117
3.2 Evaluating temporal compliance

public int evaluateTemporalCompliance()
{
  boolean containsMultipleComplianceLevels = false;
  //check if the temporal obligation has multiple threshold values in compliance function
  if(complianceFunction.getThresholdValues().size() > 0)
  {
    containsMultipleComplianceLevels = true;
  }
  //checking if the underlying activity in the temporal obligation has been completed
  if(getActivityCompletionVariableValue() == false)
  {
    if(containsMultipleComplianceLevels == false)
    {
      currentCompliance = 0;
      status = "TERMINATED";
    }
    else
    {
      if(comparator.equalsIgnoreCase("lessthanequalto"))
      {
        currentCompliance = complianceFunction.getComplianceLevel(actualPerformance);
        status = "TERMINATED";
      }
    }
if(Double.parseDouble(actualPerformance) <=
Double.parseDouble(requiredPerformance))
{
    currentCompliance = 1;
}
else
{
    //calling compliance function to get compliance level
    currentCompliance =
        complianceFunction.getComplianceLevel(actualPerformance);
}

status = "CHECKED";
}
else
{
    if(comparator.equalsIgnoreCase("lessthanequalto"))
    {
        if(Double.parseDouble(actualPerformance) <=
            Double.parseDouble(requiredPerformance))
        {
            currentCompliance = 1;
        }
        else if (containsMultipleComplianceLevels == true)
        {
            currentCompliance =
                complianceFunction.getComplianceLevel(actualPerformance);
        }
    }
    else
    {
        currentCompliance = 0;
    }
}
else if(comparator.equalsIgnoreCase("LessThan"))
{
    if(Double.parseDouble(actualPerformance) <
        Double.parseDouble(requiredPerformance))
    {
        currentCompliance = 1;
    }
    else if (containsMultipleComplianceLevels == true)
    {
        currentCompliance =
            complianceFunction.getComplianceLevel(actualPerformance);
    }
    else
    {
        currentCompliance = 0;
    }
}

status = "TERMINATED";
3.3 Evaluating compliance level in compliance function

```java
public int getComplianceLevel(String actualPerf) {
    int complianceLevel = 0;
    actualPerformance = actualPerf;
    // checking the number of threshold values in the compliance function
    for(int i = 0; i < thresholdValues.size(); i++) {
        // checking the comparator to be applied
        if(comparator.equalsIgnoreCase("lessthanequalto")) {
            if(Double.parseDouble(actualPerf) <= Double.parseDouble(thresholdValues.get(i))) {
                for(int j = 0; j < thresholdValues.size(); j++) {
                    if(i == j) // establishing compliance level
                        {complianceLevel = -(j+1);
                         i = thresholdValues.size();
                         }
                }
            } else if(comparator.equalsIgnoreCase("LessThan")) {
                if(Double.parseDouble(actualPerf) < Double.parseDouble(thresholdValues.get(i))) {
                    for(int j = 0; j < thresholdValues.size(); j++) {
                        if(i == j)
                            {complianceLevel = -(j+1);
                             i = thresholdValues.size();
                             }
                    }
            } else {
                complianceLevel = -(thresholdValues.size() + 1);
            }
        } else {
            complianceLevel = -(thresholdValues.size() + 1);
        }
    }
    return currentCompliance;
}
```
return complianceLevel;

7. References


