TEMPORAL INTERACTION CONTRACTS FOR COMPONENTS IN A DISTRIBUTED SYSTEM

A Thesis
Submitted to the Faculty
of
Purdue University
by
Omkar J. Tilak

In Partial Fulfillment of the Requirements for the Degree of
Master of Science

December 2006
Purdue University
Indianapolis, Indiana
DEDICATION

To the Loving Memory of My Late Grandparents Aniruddha Tilak and Usha Tilak
ACKNOWLEDGMENTS

I want to thank my parents whose continual love and encouragement has been an essential ingredient in the development of this thesis. Their continuous questions regarding the status of my thesis and their never-ending interest in my work and accomplishments has always kept me oriented and on track.

I would like to thank my professor and advisor, Dr. Rajeev R. Raje for giving me an opportunity to be part of the UniFrame project. His support and advice throughout the course of my graduate study and research work have helped me to successfully complete this thesis. I am grateful for his constant encouragement which made it possible for me to explore and learn new things.

I would like to thank Dr. Xukai Zou for being part of my graduate committee, and for his guidance in my course work. I would also like to thank Dr. Andrew M. Olson for his valuable suggestions throughout the research period, and for reviewing my thesis.

I would like to thank the entire staff of the Department of Computer and Information Science for their assistance during my graduate study.

I would also like to thank my cousin Mayur and his wife Sneha who have helped me a lot during my stay in the United States.

Last, but certainly not the least, I would like to thank Dada Kaka and Kaku for their constant encouragement and support towards my education.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>ABBREVIATIONS</td>
<td>ix</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Problem Definition and Motivation</td>
<td>3</td>
</tr>
<tr>
<td>1.2. Motivation</td>
<td>5</td>
</tr>
<tr>
<td>1.3. Contributions</td>
<td>6</td>
</tr>
<tr>
<td>1.4. Thesis Organization</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 2. RELATED WORK</td>
<td>8</td>
</tr>
<tr>
<td>2.1. Design by Contract</td>
<td>8</td>
</tr>
<tr>
<td>2.2. Components with Contracts</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1. Syntax Level Contract</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2. Semantic Level Contract</td>
<td>10</td>
</tr>
<tr>
<td>2.2.3. Synchronization Level Contract</td>
<td>11</td>
</tr>
<tr>
<td>2.2.4. Quantitative Level Contract</td>
<td>11</td>
</tr>
<tr>
<td>2.3. Signature and Specification Matching</td>
<td>11</td>
</tr>
<tr>
<td>2.4. Synchronization and Quality of Service Specification and Matching of Software Components</td>
<td>12</td>
</tr>
<tr>
<td>2.5. Extending CORBA (Common Object Request Broker Architecture)</td>
<td></td>
</tr>
<tr>
<td>2.6. Behavior Protocols for Software Components</td>
<td>18</td>
</tr>
<tr>
<td>2.6.1. Interaction Specification in SOFA</td>
<td>18</td>
</tr>
<tr>
<td>2.7. Formal Specification and Prototyping of CORBA Systems</td>
<td>19</td>
</tr>
<tr>
<td>2.8. DIL (Distributed Interaction Language)</td>
<td>21</td>
</tr>
<tr>
<td>2.9. Midas</td>
<td>22</td>
</tr>
<tr>
<td>CHAPTER 3. TEMPORAL INTERACTION CONTRACTS AND MATCHING ALGORITHMS</td>
<td>26</td>
</tr>
<tr>
<td>3.1. Temporal Interaction Contracts</td>
<td>26</td>
</tr>
<tr>
<td>3.1.1. States</td>
<td>29</td>
</tr>
<tr>
<td>3.1.2. Transitions</td>
<td>30</td>
</tr>
<tr>
<td>3.1.3. Events</td>
<td>30</td>
</tr>
<tr>
<td>3.2. XML representation of Temporal Interaction Contracts</td>
<td>31</td>
</tr>
<tr>
<td>3.3. Guidelines for Writing Interaction Contracts</td>
<td>31</td>
</tr>
<tr>
<td>3.4. Component Specification Matching</td>
<td>32</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.5. Data Structure Definitions</td>
<td>33</td>
</tr>
<tr>
<td>3.6. Function Definitions</td>
<td>36</td>
</tr>
<tr>
<td>3.7. Matching Algorithms</td>
<td>38</td>
</tr>
<tr>
<td>3.7.1. ‘Initialize’ Algorithm</td>
<td>41</td>
</tr>
<tr>
<td>3.7.2. ‘Calculate Time Slice’ Algorithm</td>
<td>44</td>
</tr>
<tr>
<td>3.7.3. ‘Calculate Interaction Trace’ Algorithm</td>
<td>48</td>
</tr>
<tr>
<td>3.7.4. ‘Calculate Intersection Trace’ Algorithm</td>
<td>50</td>
</tr>
<tr>
<td>3.7.5. ‘Contains’ Algorithm</td>
<td>52</td>
</tr>
<tr>
<td>3.7.6. ‘Equivalent under Compatibility Constraint’ Algorithm</td>
<td>54</td>
</tr>
<tr>
<td>3.7.7. ‘Equivalent under Replaceability Constraint’ Algorithm</td>
<td>56</td>
</tr>
<tr>
<td>3.7.8. ‘Check Partial Compatibility’ Algorithm</td>
<td>58</td>
</tr>
<tr>
<td>3.7.9. ‘Check Complete Compatibility’ Algorithm</td>
<td>59</td>
</tr>
<tr>
<td>3.7.10. ‘Check Partial Replaceability’ Algorithm</td>
<td>60</td>
</tr>
<tr>
<td>3.7.11. ‘Check Complete Replaceability’ Algorithm</td>
<td>61</td>
</tr>
<tr>
<td>3.8. Control Flow for Determining Partial Compatibility</td>
<td>61</td>
</tr>
<tr>
<td>3.9. Control Flow for Determining Partial Replaceability</td>
<td>63</td>
</tr>
<tr>
<td>3.10. Control Flow for Determining Complete Compatibility</td>
<td>63</td>
</tr>
<tr>
<td>3.11. Control Flow for Determining Complete Replaceability</td>
<td>67</td>
</tr>
<tr>
<td>3.12. Conclusion</td>
<td>67</td>
</tr>
<tr>
<td>CHAPTER 4. CASE STUDY</td>
<td>68</td>
</tr>
<tr>
<td>4.1. TIC Prototype (TICP) and Experimental Setup</td>
<td>68</td>
</tr>
<tr>
<td>4.2. Telephone System</td>
<td>69</td>
</tr>
<tr>
<td>4.2.1. Temporal Interaction Contract for ‘Telephone’ component</td>
<td>70</td>
</tr>
<tr>
<td>4.2.2. Temporal Interaction Contract for ‘Dialer’ component</td>
<td>73</td>
</tr>
<tr>
<td>4.2.3. Compatibility Analysis</td>
<td>75</td>
</tr>
<tr>
<td>4.2.4. Modified TIC and Recomputation of Compatibility</td>
<td>80</td>
</tr>
<tr>
<td>4.2.5. Replaceability Analysis</td>
<td>84</td>
</tr>
<tr>
<td>4.3. E-commerce System</td>
<td>92</td>
</tr>
<tr>
<td>4.3.1. Temporal Interaction Contract for ‘Shopping Cart’ component</td>
<td>96</td>
</tr>
<tr>
<td>4.3.2. Temporal Interaction Contract for ‘Credit Card’ component</td>
<td>97</td>
</tr>
<tr>
<td>4.3.3. Temporal Interaction Contract for ‘Customer’ component</td>
<td>99</td>
</tr>
<tr>
<td>4.3.4. Compatibility and Replaceability Analysis for E-commerce System</td>
<td>101</td>
</tr>
<tr>
<td>4.4. TIC Simulation</td>
<td>104</td>
</tr>
<tr>
<td>4.4.1. ‘Telephone’ and ‘Dialer’ component</td>
<td>105</td>
</tr>
<tr>
<td>4.4.2. Modified ‘Telephone’ and ‘Dialer’ component</td>
<td>106</td>
</tr>
<tr>
<td>4.5. Summary</td>
<td>108</td>
</tr>
<tr>
<td>CHAPTER 5. CONCLUSION AND FUTURE WORK</td>
<td>109</td>
</tr>
<tr>
<td>5.1. Conclusions of the Research</td>
<td>109</td>
</tr>
<tr>
<td>5.2. Overview of Temporal Interaction Contracts</td>
<td>110</td>
</tr>
<tr>
<td>5.3. Contributions of this Thesis</td>
<td>111</td>
</tr>
<tr>
<td>5.4. Future Work</td>
<td>111</td>
</tr>
<tr>
<td>5.5. Summary</td>
<td>112</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>113</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Appendix A. XML Specification of TICs</td>
<td>116</td>
</tr>
<tr>
<td>Appendix B. Source Code</td>
<td>141</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1 InitialTimeSlice for TIC T₁</td>
<td>43</td>
</tr>
<tr>
<td>Table 3.2 InitialTimeSlice for TIC T₂</td>
<td>43</td>
</tr>
<tr>
<td>Table 3.3 InitialTimeSlice for TIC T₃</td>
<td>44</td>
</tr>
<tr>
<td>Table 3.4 TimeSlice for TIC T₁</td>
<td>46</td>
</tr>
<tr>
<td>Table 3.5 TimeSlice for TIC T₂</td>
<td>47</td>
</tr>
<tr>
<td>Table 3.6 TimeSlice for TIC T₃</td>
<td>47</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3.1 Temporal Interaction Contract of Electric Bulb</td>
<td>28</td>
</tr>
<tr>
<td>Figure 3.2 Temporal Interaction Contract T₁ (Graphical Representation)</td>
<td>38</td>
</tr>
<tr>
<td>Figure 3.3 Temporal Interaction Contract T₂ (Graphical Representation)</td>
<td>39</td>
</tr>
<tr>
<td>Figure 3.4 Temporal Interaction Contract T₃ (Graphical Representation)</td>
<td>40</td>
</tr>
<tr>
<td>Figure 3.5 ‘Contains’ Constraint</td>
<td>54</td>
</tr>
<tr>
<td>Figure 3.6 Determine Partial Compatibility</td>
<td>62</td>
</tr>
<tr>
<td>Figure 3.7 Determine Partial Replaceability</td>
<td>64</td>
</tr>
<tr>
<td>Figure 3.8 Determine Complete Compatibility</td>
<td>65</td>
</tr>
<tr>
<td>Figure 3.9 Determine Complete Replaceability</td>
<td>66</td>
</tr>
<tr>
<td>Figure 4.1 A Typical Telephone System</td>
<td>70</td>
</tr>
<tr>
<td>Figure 4.2 TIC of ‘Telephone’ Component (Graphical Representation)</td>
<td>71</td>
</tr>
<tr>
<td>Figure 4.3 TIC of ‘Dialer’ Component (Graphical Representation)</td>
<td>73</td>
</tr>
<tr>
<td>Figure 4.4 Modified TIC of ‘Telephone’ Component (Graphical Representation)</td>
<td>81</td>
</tr>
<tr>
<td>Figure 4.5 Modified TIC of ‘Dialer’ Component (Graphical Representation)</td>
<td>82</td>
</tr>
<tr>
<td>Figure 4.6 TIC of ‘Telephone1’ Component (Graphical Representation)</td>
<td>85</td>
</tr>
<tr>
<td>Figure 4.7 Modified TIC of ‘Telephone’ component (Bold font indicates changes)</td>
<td>89</td>
</tr>
<tr>
<td>Figure 4.8 Modified TIC of ‘Telephone1’ Component (Bold font indicates changes)</td>
<td>90</td>
</tr>
<tr>
<td>Figure 4.9 Class Diagram of an E-commerce System</td>
<td>94</td>
</tr>
<tr>
<td>Figure 4.10 Class Diagram of an E-commerce System (continued)</td>
<td>95</td>
</tr>
<tr>
<td>Figure 4.11 TIC of ‘Shopping Cart’ Component (Graphical Representation)</td>
<td>96</td>
</tr>
<tr>
<td>Figure 4.12 TIC of ‘Credit Card’ Component (Graphical Representation)</td>
<td>98</td>
</tr>
<tr>
<td>Figure 4.13 TIC of ‘Customer’ Component (Graphical Representation)</td>
<td>100</td>
</tr>
<tr>
<td>Figure 4.14 ‘Credit Card’ Component With Its ‘Environment’</td>
<td>101</td>
</tr>
<tr>
<td>Figure 4.15 TIC For the ‘Environment’</td>
<td>103</td>
</tr>
</tbody>
</table>
ABBREVIATIONS

TIC – Temporal Interaction Contract
TICP – Temporal Interaction Contract Prototype
TICS – Temporal Interaction Contract Simulation
ABSTRACT

Tilak, Omkar J. M.S., Purdue University, December, 2006. Temporal Interaction Contracts for Components In A Distributed System. Major Professor: Rajeev Raje.

Software realization of large-scale Distributed Computing Systems (DCS) is achieved through the Component-based Software Development (CBSD) approach. Software executing on distributed systems consists of many autonomous components which interact with each other to coordinate the activity. The need for such coordination along with requirements such as heterogeneity, scalability, security, and availability, considerably increases the complexity of code in distributed systems. Simplifying distributed component-based software development requires abstractions that enable separate design and specification of component interactions and the components themselves. This thesis depicts a formal method to specify component interactions involving temporal constraints. Based on the knowledge of component interactions, various types of interaction compatibility and replaceability classes are defined. A prototype reflecting various ideas proposed is implemented and experimented with. The experimental analysis using the prototype indicates the benefits of the component interaction specifications mentioned in this thesis.
CHAPTER 1. INTRODUCTION

In the past, computer applications were designed to run on a single computer operating in isolation. But with the advent of networking, new technologies for high bandwidth, wireless data communications and the availability of cheaper and more powerful computers; more and more applications are being designed to run on networks of cooperating computers. This approach has lead to the development of distributed computing systems (DCS). The inherent distribution of DCS has meant that components (software and hardware) can be physically as well as logically distributed. This aspect makes the software design of a DCS a challenging task.

The most widely used paradigm for realizing the software of DCS is Component Based Software Development (CBSD) [COU01]. CBSD can be defined as the process of building large and complex software systems by integrating previously developed and deployed software components [SZY03]. The fundamental principle that lies at the heart of CBSD is reuse. It is proposed that software systems should be assembled through a reuse of its constituent components rather than writing them from scratch. This approach offers a lot of benefits. It increases the automation and productivity in the software development by streamlining the development cycle. It also contributes to a reduced development and testing time, and thus, reduces the software development cost. It improves the quality and reliability of the generated software. It reduces the maintenance cost associated with the support and updates of software. A comprehensive overview of advantages of CBSD approach is provided in [PRE97].
The CBSD approach could be used to assemble DCS using previously developed and deployed software components that are distributed across the network. These software components may be independently developed and deployed possibly by different developers without a priori knowledge about the system that they may be used for and other software components with which they may interact. Although there are many views on what constitutes a software component, the consensus is that a component is characterized by the services it provides and requires, typically via interfaces. Many component models including Darwin [23], Wright [2], and SOFA [40] follow this paradigm. Also, the software components that are integrated to form DCS could belong to different distributed computing models such as Java Remote Method Invocation (RMI) [GER04], Distributed Component Object Model (DCOM) [DAV02], Common Object Request Broker Architecture (CORBA) [HOU98], and .NET [MIC03]. Thus heterogeneity is an inherent feature of developing a DCS. So there is a need to take care of the heterogeneity that may arise in the creation of DCS. A comprehensive overview of challenges involved in CBSD approach is provided in [CRN01].

The interfaces to components are described in terms of signatures of methods available on the interface. This allows reasoning about the correctness of component composition at the level of syntactical type correspondence. However, when considering composition of components obtained from different sources (vendors, development teams), such a specification may not be sufficient. A specification of the component’s behavior is essential, describing the correct mode of the usage of the services provided by the component, as well as the dependence among interactions occurring on the individual interfaces of the component. So an important requirement in building DCS using the CBSD approach is the specification of interactions supported by a software component. This interaction specification describes the sequence in which methods supported by a component should be accessed. Thus, assembling a distributed
application requires abstractions that enable a separate design of component interactions.

Applying component behavior specification methods to Component Based Software Development (CBSD) can yield significant benefits, either as the option to employ a tool to verify the specification of the system being designed, or the possible CASE support in developing the component implementation. A CASE tool may utilize the behavior specification in deriving the initial design, as well as in generating skeleton code for operation implementations. Also, specification of component interactions enables system developers to customize and modify components as required by an application. At the same time, the interaction specification may be reused with different applications. Thus, a separation of component implementation and the interaction behavior may facilitate the design and implementation of the distributed systems.

The component behavior abstraction presents an interface representing the interaction policy while hiding the details of the mechanisms used to implement the policy. It should be observed that an interaction policy determines the interface, not the implementation; hardware and other system variables determine which protocol is used to implement a given interaction policy. The component behavior specification abstracts these implementation details and specifies component behavior in a platform independent manner. Thus, a component interaction specification facilitates construction of a distributed system.

1.1. Problem Definition and Motivation

A vital issue in the correct use of commercial-off-the-shelf (COTS) components is the proper understanding of their functionality, quality attributes, and ways of operation. The component specification should be formal and without any ambiguous or inconsistent statements. Equipped with the component
specification clearly defining the basic aspects of component use, such as operation signatures and operating platforms, this thesis will provide a formal means for addressing the behavioral interoperability issues in component-based system design. This thesis proposes an approach for component specification that augments existing component specification with the capability of specifying component interactions. This approach uses temporal ordering to define sequencing constraints between component operation invocations.

The first part of the thesis deals with specifying component interaction contracts. This thesis proposes the specification of component interactions with temporal constraints indicating when an operation can be invoked. It focuses in particular on problems related to compatibility and replaceability analysis in the context of interactions.

Component specifications can be divided into four different levels [BEU99]. **Syntactic level** gives information about the syntax of the interfaces of the software components. This facilitates the matching of the type of the parameters of the provided services against required services. **Semantic Level** specifications describe the component’s dynamic behavior. It makes use of pre- and post-conditions to describe semantics of the method accesses. **Synchronization Level** describes the synchronization policies used by the components to synchronize multiple clients accessing the component’s interfaces. **Quality of Service (QoS) Level** describes the values of the pre-specified QoS parameters, such as turn around time and throughput. It also provides information about the dependency of the QoS parameters on the environment variables and usage patterns. The focus of this thesis is to study component behavior specification; so the component specifications described in this thesis can be thought of as a part of the semantic contract.
The second part of the thesis constitutes defining the different types of matches with respect to replaceability and compatibility in the context of temporal interaction contracts. Replaceability refers to the ability of two components to replace each other within DCS. Compatibility is the ability of two components to interoperate, communicate and cooperate with each other when brought together to form a distributed system.

The specific objectives of this thesis are as follows:

1. Provide a mechanism for the specification of the component behavior.
2. Define a set of operators for comparing two component behavior specifications.
3. Develop algorithms to implement these component behavior comparison operators.
4. Develop a prototype to validate the matching techniques developed.

1.2. Motivation

The motivation for providing mechanisms for the specification of the component’s interaction behavior and for matching the component specifications is as follows:

A key feature of component-based software engineering is that it allows the relatively independent development of components and the construction of a software system using components developed by third-parties [CRN01]. Underlying this independence is the components’ interface definitions that serve as contracts between service provider components and service consumer components. A service consumer component will be able to use the services of a provider component based on its interface definition without knowing its implementation details. On the other hand, the service provider component can
be implemented based on its interface definition without knowing the potential users or consumers. As such, the component interface definition plays a vital role in ensuring the compatibility between the components of a composite system [GAR95].

Commercial IDLs (including CORBA IDL) primarily address the signature aspects of software component interfaces, i.e., the names, parameters and data types of the provided operations. They do not provide support for capturing the semantic or behavioral aspects of a component, including its usage, capabilities and interaction behavior.

To provide a sound support for studying the component interoperability, unambiguous descriptions, clearly stating the semantics, qualities and interactions of the services provided and required by a software component, are needed. In particular, the component interaction specification describes the rules that govern the interactions of the component, i.e., the valid sequences of its service/operation invocations. These include the way in which a component provides its services and the order in which its operations are to be invoked so as to facilitate the proper use of its services.

1.3. Contributions

Contracts play an important role in the creation of a high confidence DCS [BEU99]. The thesis provides a formal approach for the specification of component interactions. The thesis also provides a set of matching criteria that enables users to match their requirements against the component specifications and to match two component specifications. The contributions of the thesis are:

1. Definition of the criteria for the matching of the software components behavior contracts: The thesis provides a set of matching criteria that is used to match the contracts at the semantic level.
2. Validation of the proposed mechanism for the matching of the contracts: The thesis validates the proposed mechanisms by specifying the contracts for a prototype system and matching the specifications in order to determine if the components can interact or not.

1.4. Thesis Organization

This thesis is organized into five chapters. An introduction, along with the problem definition and motivation, objectives, and the contributions are provided in this chapter. Chapter 2 discusses the background and related work of this thesis. Chapter 3 gives the design and implementation details of the temporal interaction contract specification along with algorithms to check compatibility and replaceability between temporal interaction contracts. Chapter 4 describes a case study for experimental analysis of these concepts along with a prototype implementation. Chapter 5 provides the conclusion of this research work, the possible future enhancements, and the summary of this thesis.
CHAPTER 2. RELATED WORK

In the previous chapter, a brief introduction of the thesis is presented along with the problem definition, the thesis objectives and the contributions of this thesis. This chapter provides a discussion of the background and few of the related research works of the thesis. The next section talks about the ‘Design by Contract’ that provides the foundation for this research.

2.1. Design by Contract

The main challenge in the development of a DCS is how to make a robust and correct distributed system using heterogeneous software components. The robustness of a software system refers to the ability of the system to handle abnormal situations. CBSD (Component Based Software Development) is a promising approach for the development of a DCS and reliability particularly becomes important in CBSD because of the concept of reusability of the software components. Hence, one of the major issues involved in CBSE is how an application developer trusts a particular software component.

To make sure that a DCS performs correctly, there is a need for a systematic approach for specifying and implementing software components. For this purpose, Meyer [MEY92] introduced a theory called Design by Contract. In this theory, a software system is viewed as a set of communicating components whose interaction is based on precisely defined specifications of the mutual obligations, called Contracts.
The Design by Contract theory provides the following benefits:

1. A better understanding of the component-based software development method.
2. A systematic approach to building bug-free DCS.
3. An effective framework for debugging, testing and, more generally, quality assurance.
5. Better understanding and control of the inheritance mechanism.
6. A technique for dealing with abnormal cases, leading to a safe and effective language construct for exception handling.

In Design by Contract theory, the first goal is to define the functionality of each software element as precisely as possible, in order to ensure that it does what is indicated by the specification/contract. These contracts govern the interactions of the software components with its environment consisting of the execution environment and/or other software components.

2.2. Components with Contracts

Based on the theory of Design by Contract, [BEU99] defines a general model of software contracts and shows how existing mechanisms could be used to turn traditional components into contract aware ones. [BEU99] asserts that there is a need for a way of determining whether a given component can be used in a particular context. It indicates that this information would take the form of a specification that tells us what the component does, without entering into the details of how. The specifications should provide parameters against which the components can be verified and validated, thus, providing a contract between the component and its users.

Four classes of contracts are defined in the context of software components [BEU99]: syntax, behavioral, synchronization and quantitative. This
classification is based on the increasing negotiable properties of the various levels of contract. The hierarchy based on the increasing negotiability of the different levels of contracts is as follows: 1) Syntactic, 2) Semantic, 3) Synchronization, and 4) Quantitative, with syntactic level contract at the bottom of the hierarchy.

2.2.1. Syntax Level Contract

The first level contract, i.e., the syntactic level contract, takes the form of specifying the operations that a component perform, the operation’s input and output parameters and the possible exceptions that might be raised during the execution of the operation. The static type checking and the dynamic type checking forms an important part of syntax level contract. [BEU99] provides a mechanism to verify that all clients use the component interface properly; static type checking performs the verification at the compile time whereas the dynamic type checking performs the verification at the runtime.

2.2.2. Semantic Level Contract

The syntactic specification of a component’s operation provides information about how the operation may be invoked by other components but it does not define precisely the effect of the execution of the operation. For instance, the signature of adding two integers and subtracting two integers may be same, with two integer input parameters and an integer return value, but, they cannot be distinguished behaviorally with the information provided by the syntactic contract. Thus, there is a need for describing the behavior of the operations supported by the component. The behavioral property of a component’s property can be specified using Boolean assertions in the form of the pre- and post- conditions.
2.2.3. Synchronization Level Contract

The synchronization level contract provides the behavior of the component in terms of synchronization between the concurrent method calls. A contract at this level specifies the ways in which the component serves multiple clients. In a single server and multiple clients environment, the synchronization level contract guarantees to each client that whatever the other client’s request may be, the requested service will be executed according to its specifications, but in a non-deterministic manner.

2.2.4. Quantitative Level Contract

The Quantitative level contract helps specifying the QoS attributes of the component’s services, such as the turn-around time, throughput, maximum response delay, average response, and the quality of the result. The negotiability at this level of contract is the highest.

This thesis uses the idea of multi-level contracts and extends it to describe the component interactions and provide a matching of the interaction specifications.

2.3. Signature and Specification Matching

The research described in [ZAR96] presents a way to compare two components to help realize more of the potential of software libraries. It uses semantic information associated with the software components to match the requirements of the clients. Each component stored in a component library has a signature (type information) and a specification (behavioral information). It is claimed that in all the cases the components’ behavior plays an important role and matching of their specifications becomes an essential step in order to determine whether a particular relation holds among the components.
For matching the component’s specifications, the research takes three parameters into consideration:

- The kind of information used to describe the components – a component abstraction/description may vary from a textual description, a control- or Data-flow graphs to the signatures to the semantic information of the operations of the component.
- The granularity of the components – components vary in size from the functions, the modular collections of functions to the stand-alone software systems.
- The degree of relaxation of the match – for each kind of match the research provides exact and relaxed sets of matches.

This research project uses ML as the component implementation language and Larch/ML as the specification language. Larch/ML is a Larch Interface Language for Modula and provides a mechanism for specifying the syntax of the component operations and the behavior of the component.

This thesis extends this specification matching by providing a matching of the interaction level in the multi-level specification.

2.4. Synchronization and Quality of Service Specification and Matching of Software Components

The research described in [KUM04] extends the work described in [ZAR97] by providing a mechanism for the formal specification of the component’s properties at the synchronization and the Quality of Service levels. Research aims to identify types of matches applicable to different levels and provide a mechanism to match the specifications of component in order to determine substitutability or compatibility relationships between them. The research defines different types of matches with respect to substitutability and
compatibility. Substitutability refers to the ability of two components to replace each other within a system. Compatibility is the ability of two components to interoperate, communicate and cooperate with each other when brought together to form a system. The matching of component’s specifications is done at all the four levels with respect to both the substitutability and the compatibility, and it can be divided into two or more categories for each of the criteria. This thesis uses TLA+ [LAM94] language to specify the synchronization policies, and the synchronization aspects of a software component.

The synchronization level contract for a component consists of following predicates:

1. Name of the component: Component to which this contract applies.
2. Name of the component interface: Specific interface of the component for which this contract applies.
3. Name of the synchronization policy: Synchronization policy used by the component.
4. Synchronization policy implementation method: Technique used to implement this synchronization policy.
5. Synchronization pre-condition: Pre-condition with respect to synchronization behavior of the component.
6. Synchronization invariant: Invariant with respect to synchronization behavior of the component.
7. Synchronization action: Action taken by the component with respect to the synchronization behavior.
8. Synchronization post-condition: Post-condition with respect to synchronization behavior of the component.

The QoS level contract for a component provides the formal representation of the following properties of the QoS parameters:

1. Name of QoS parameter: Specifies name of the QoS parameter.
2. Domain dependency / Domain of usage: Indicates if this QoS parameter is domain dependent or not.

3. Composability: Indicates if this QoS parameter can be used during component composition process.

4. Value type: Indicates the type of the value of QoS parameter (e.g. integer, float etc).

5. Static / Dynamic: Indicates if the QoS parameter is constant or varies over a period of time.

6. Aliases: Indicates equivalent names of this parameter.

7. Nature of QoS parameter: Indicates the category of this QoS parameter.

8. Component / Method: Indicates if this QoS parameter is associated with the whole component or a particular method of the component.

The following matching criteria are defined for matching synchronization policies of components:

1. Generalized exact match: In generalized exact match, there exist a logical equivalence relationship between the synchronization policies of the components and the synchronization pre- and post-conditions of the methods of the components.

2. Generalized relaxed match: In generalized relaxed match, there exist a logical implication relationship between the synchronization policies of the components and the synchronization pre- and post-conditions of the methods of the components.

3. Specialized exact match: The exact match for the specialized case, is the same as exact match for the generalized exact case except that it also checks if the implementation method (e.g. synchronization using semaphore, synchronization using monitors) of the synchronization policy match or not.
4. Specialized relaxed match: The relaxed match for the specialized case is the same as that of the generalized except that the implementation method for the synchronization policy has to be matched for the query component QC and the component C.

The following matching criteria are defined for matching QoS contracts of components:

1. Exact match: A component is said to be an exact match of a query component, if the value of the QoS parameters of the component is equal to or better than that of the query component. “Better” is one of the logical operators, greater than equal to, equal to, or less than equal to. It is defined differently for different QoS parameters and it depends on the nature of the QoS parameter.

2. Relaxed match: A component is said to be a relaxed match of a query component, if value of the QoS parameters of the component is close to the value of the parameter for the query component. A “close” value, for some of the parameters, is defined by the user of the component based on the requirements of the user. The user gives the measure of the deviation that is allowed in the values of the parameters while performing the matching.

Using these matching criteria, the following QoS matches can be performed on the component contracts:

1. Exact match (Match all the parameters): Matches each of the QoS parameters from the component’s QoS contract with that of the query component for a “better” value of the parameter.

2. Exact match (Match the specified parameters): Matches only the specified parameters of the components QoS contract with that of the query component for a “better” value of the parameter.
3. Relaxed match (Match all the parameters): Matches all the QoS parameters from the component's QoS contract with that of the query component for a "close" value of the parameter.

4. Relaxed match (Match the specified parameters): Matches only the specified parameters of the components QoS contract with that of the query component for a "close" value of the parameter.

In summary, the research provides matching techniques for synchronization and QoS level in the component contract. However, it does not take into consideration the timing constrains associated with the component interaction. This thesis extends specification matching by providing a matching of component specifications in the context of temporal interaction constraint.

2.5. Extending CORBA (Common Object Request Broker Architecture) Interfaces with Protocols

The research described in [CAN01] describes an extension of the CORBA IDL (Interface definition Language) for describing object service interactions, aimed at the automated checking of interaction interoperability between CORBA objects in open component-based environments.

The research concentrates on the interoperability of reusable components at the interaction level which deals with the relative order among their incoming and outgoing messages, and their blocking conditions. An extension to CORBA IDL is proposed that allows the description of the interactions of CORBA objects, in addition to the static description of the object services (i.e., method signatures) provided by the standard CORBA IDL facilities. This approach enriches IDLs with information about the way objects expect their methods to be called, how they use other objects' methods, and even interaction semantic aspects of interest to users, deployers, and implementers of objects.
Interactions are described using a sugared subset of the polyadic $\pi$-calculus [MIL93], and defined separately from the IDLs. The polyadic $\pi$-calculus allows tuples and compound types to be sent along channels. The $\pi$-calculus has proved to be a very expressive notation for describing the dynamic behavior of objects in applications with changing topologies (as those that live in open systems). In this sense, it is more appropriate than other process algebras such as CCS or CSP. In this way current repositories can be easily extended to account for this new information and to manage it. In addition, having this interaction information available at run time also allows defining dynamic compatibility checks in open and extensible applications.

The $\pi$-calculus can be used to express component interactions that make use of operation parameters and return values when considering alternatives, which is an important issue concerning the expressiveness of interaction specifications, and that represent some advantages over other approaches, such as those based on state-machines. Another extra benefit of this formal notation is that it allows specifying, in addition to the specific interaction information, some of the details of the object's internal state and semantics that are relevant to its potential users, while hiding those which can be left open to possible implementations. In this sense, $\pi$-calculus also offers this advantage over other formal notations for describing mere component interaction information, such as Message Sequence Charts (MSCs), for instance.

The use of polyadic $\pi$-calculus to specify component interactions requires a high degree of mathematical rigor. Also, system developers have to learn polyadic $\pi$-calculus in detail to understand the specification. The research presents no formalism to facilitate matching of component specifications. The aim of this thesis is to present component interactions using an easy-to-understand and comprehensive formalism.
2.6. Behavior Protocols for Software Components

The research described in [PLA98] proposes a means to enhance an architecture description language with a description of the component behavior. A notation used for this purpose depicts the “interplay” on the component’s interfaces and reflect step-by-step refinement of the component’s specification during its design. In addition, the notation allows for formal reasoning about the correctness of the specification refinement and also about the correctness of an implementation in terms of whether it adheres to the specification or not. The research proposes to employ behavior protocols which are based on a notation similar to regular expressions. As a proof of the concept, the behavior protocols are used in the SOFA (SOFtware Appliances) [PLA98] architecture description language at three levels: interface, frame, and architecture. By using SOFA tools, the designer can verify the adherence of a component’s implementation to its specification at run time, while the correctness of refining the specification can be verified at design time.

2.6.1. Interaction Specification in SOFA

SOFA description abstracts from a particular component model and most of the ADL-dependent details such as name spaces, typing rules, etc. The SOFA model relies on an abstract component – called agent, where the interface ties of components become connections among agents, method calls on interfaces turn into events on connections, and behavior of the component is modeled via the event sequences (traces) on the connections of the agent representing the component. The agent behavior can be approximated and represented by regular expression-like “protocols”. The analysis of these regular expressions allows to reason about the component cooperation at the assembly time and at the run time.

An agent can be primitive or composed. A primitive agent is an agent which does not contain any other agent and all of its connections are external.
An agent handles a sequence of events. An agent can perform various activities such as emitting events, absorbing events, and processing internal events. To transmit events to each other, agents communicate via peer-to-peer bidirectional connections. An agent can communicate through a finite number of connections. This implies that an agent communicates with a finite number of other agents. The number of finite connections will vary depending upon the system architecture.

An agent can handle only one event at a time. An agent is assumed to emit an event only if its counterpart on the connection is prepared to accept it. Transmission delays are ignored or assumed to be negligible. A sequence of events handled by an agent is represented as an event trace. The set of all possible sequences of event handling of an agent in any run is referred to as the behavior of that agent.

To be able to formulate the contract at a higher level of abstraction, it is assumed every trace representing a particular communication between an agent and its environment is an interleaving of two logical parts provision and requirement. In this scenario, an agent provides (offers) services to its environment. Similarly, an agent can require some services from its environment.

SOFA specification captures component interactions by using agents and events framework. However, it does not provide any formal mechanism to match component specifications based on the time constraints associated with component interactions. This thesis focuses on specification of time constraints involved in component interactions.

2.7. Formal Specification and Prototyping of CORBA Systems

The research described in [BAS99] extends the CORBA interface definition of distributed objects by a behavioral specification based on high-level
Petri nets [PET81]. This technique allows specifying in an abstract, concise and precise way the behavior of CORBA servers, including internal concurrency and synchronization. This approach also enables to early prototyping and testing of a distributed object system as soon as the behaviors of individual objects have been defined.

Petri nets have been studied for a long time as a mathematical formalism for the modeling of concurrent systems. Petri net models a system as a set of state variables called places and a set of state-changing operators called transitions. The state of the system is described as a distribution of information elements (called tokens) in the net’s places; this distribution is called the marking of the net. In basic Petri nets, tokens are dimensionless entities modeling only conditions. Several dialects of Petri nets (called high-level Petri nets) allow tokens to carry information and to manipulate this information at the occurrence of transitions. High-level Petri nets differ from the basic Petri nets by the nature of the inscriptions attached to the net elements, and by their object-oriented structure.

- Tokens represent tuples of typed values. The arity of a token is the number of values it holds, and tokens of zero-arity are thus the “basic” tokens used in conventional Petri nets.
- Places are defined to hold tokens of a certain Token-type. Thus, all tokens stored in one place have the same Token-type and arity. A place holds a multiset of tokens. Thus, a given token may be present several times in the same place.
- Each arc is inscribed by a tuple of variables. The arity of an arc is the number of variables associated to it. The arity of an arc is same as the arity of the Token-type of the place it is connected to, and the type of each variable is deduced from this Token-type. The multiplicity of an arc is the number of identical tokens that will be processed by the firing of a transition associated to this arc.
Transitions have a precondition (a Boolean expression of their input variables) and an action, which may use any service allowed for the types of their input or output variables. The scope and type of each variable of an arc is local to the transition the arc connects to.

In summary, the research gives a technique to represent component interactions using high-level Petri nets. However, it does not deal with matching component specifications for compatibility and replaceability analysis. Also, it does not take into consideration, the time constraints associated with component interactions. One major goal of this thesis is to provide operators to perform component compatibility and replaceability analysis based on temporal component interaction specifications.

2.8. DIL (Distributed Interaction Language)

DIL [STU96] provides a protocol abstraction which encapsulates the component interaction protocol behavior. Much as the object-oriented programming realizes the abstract data types by enabling the publication of a component's interface while hiding its implementation, protocol abstraction presents an interface representing the interaction policy while hiding the details of the coordination mechanisms used to implement the policy.

In DIL, each protocol describes a customized response to events within the system. Distributed interaction policies are implemented by specifying events defining component interactions. Based on the interaction policy, DIL protocols may dynamically modify the system behavior: for example, in response to communication events, a protocol may add or remove messages, record or replace state, or halt the execution of a component. Events may also be programmer defined. A combination of system and programmer-defined events together defines an interaction policy; the protocol provides an implementation
for these events and, therefore, an implementation for the policy. The protocol may be modified without affecting the events comprising the policy.

In DIL, a protocol is defined in three parts: parameters which support sharing of initialization data throughout a protocol, protocol operations which define operations performed on the protocol as a whole, and role definitions which define local behavior and event responses. Protocol parameters are a set of global values which are shared by all participants in the protocol. DIL allows a finite number of components to participate in a protocol at any time. Parameters enable further customization of protocols. For example, a replication protocol may use a parameter to specify the number of replicas in the protocol.

To install a protocol in DIL, a protocol instance is created. This instance operates independently of all other protocol instances. Protocol operations may be invoked on the protocol instance to assign roles to components in the system. Roles in a protocol are relative to a particular protocol instance. The independence of each protocol instance is essential to keep protocol design modular. This implies that protocols may be written independently and then composed at run-time.

In summary, DIL allows modular specification of interaction policies in distributed systems. However, DIL focuses mainly on describing properties of interaction like atomicity and reliability. So, DIL describes component interactions at a fairly low level of abstraction. This thesis will focus of component interactions specification at a high level of abstraction. This will make the specification more readable and easy to understand.

2.9. Midas

Midas [PRY00] is used to define interactions between components. Midas definitions are compiled into runtime support code that makes use of the
underlying transport framework. This framework hides the platform-specific transport API and defines transport protocols as compositions of lightweight components. This componentization of the transport protocol allows the designer to select the most appropriate protocol for each binding and insert additional functionality, such as compression or encryption, above existing protocols.

Midas supports both the design-time analysis of component interaction styles and the construction of components and systems that make use of those interaction styles. The design is supported by including specifications of the component interaction that can be checked mechanically for the deadlock or violation of user-defined constraints. Construction is supported by translating Midas definitions into implementation language constructs that provide the “glue” to connect components using the interaction style whether they are within the same address space or across the network. Midas implements strict separation of concerns: programmers can select combinations of mechanisms within a binding to achieve the behavior required for their particular context. Moreover, the Midas language links the design and construction phases by generating objects that can be inserted into bindings to check that components interacting over those bindings conform to the interaction protocol specified by the Midas definition.

Midas views a component to be a reusable element of a distributed program that encapsulates state and/or behavior behind a strict interface comprised of the roles that the component may take in various interaction protocols. A component can act as either a service provider or service client in each of the interactions in which it plays a role. Roles are represented by named, typed interaction endpoints. Interaction endpoints provide the distribution transparency to the component implementation: the programming interface to the interaction protocol is identical whether the far end of a binding is within the same address space, on the same machine or on a remote machine. Of course,
performance of local and remote interactions will differ, as will reliability, depending on the transport protocol selected for the binding.

An interaction between two communication endpoints, service and client, can be defined in terms of asynchronous messages. An interaction consists of:

- The set of asynchronous messages accepted by the service (the server-side message interface).
- The set of asynchronous messages that the service requires the client to accept (the client-side message interface).
- The programming abstractions through which the client and service view the interactions (the client- and service-side programming interfaces), including the synchronization of component threads at those endpoints.

The messages define the application-layer protocol by which components communicate over a binding. A service guarantees to react meaningfully to messages received from a client as long as those messages are in the set of messages accepted by the service at that point in time and as long as the client reacts meaningfully when the service sends messages back to the client.

A number of interaction styles can be implemented using Midas. These styles along with their description are given below:

1. LongSlot - The modeling of Midas endpoints with FSP (Finite Sequential Process) and their implementation in Java.
2. Slot - Uses generic interaction types.
3. Mport - Describes how to specify the queuing of messages for a delivery to the receiving component.
4. Func - Describes synchronization between clients and the service.
5. Event - Describes one-to-many communication. Shows how clients can selectively enable and disable messages from a service. Shows how messages sent from the service to client by the volition of the service
rather than in response to a client can cause synchronization errors, and that these errors can be detected by the model checker.

6. Attribute - Shows how the service endpoint of an interaction style can choose a subset of its clients to which to send messages and introduces interaction-specific properties defined by the designer.

In summary, Midas describes component interactions in distributed systems. However, Midas does not provide any formalism to specify time constraints. This thesis aims to define time-constrained component interactions using component interface methods.

In this chapter an overview of the existing component interaction specification mechanism was presented. Some of the drawbacks of the existing component interaction specification mechanisms were summarized. This thesis provides a component interaction specification mechanism in the context of temporal constraints. The following chapter gives the details about the temporal interaction contracts along with algorithms to perform matching of temporal interaction contracts.
CHAPTER 3. TEMPORAL INTERACTION CONTRACTS AND MATCHING ALGORITHMS

In the previous chapter, related work of this thesis is presented. This chapter presents description of Temporal Interaction Contracts along with algorithms to perform compatibility and replaceability analysis in the context of component interaction specifications. Compatibility and replaceability analysis together constitutes component specification matching activity in the context of temporal interaction contracts.

3.1. Temporal Interaction Contracts

The interaction behavior of each component in the DCS is represented by a Temporal Interaction Contract. Every component in the system is assumed to have the following properties:

1. The component responds to external events (i.e., events occurring outside the context of the component).
2. The component lifecycle can be modeled as a progression of states, transitions and events.
3. The current behavior of the component depends on its past behavior.

Every component has a lifetime. On its deployment, a component is born, and on its destruction, it ceases to exist. In between, a component may act on other components by sending messages, as well as be acted on by being the target of messages. In many cases, these messages will be simple, synchronous method invocations. For example, a Customer component might invoke an operation getAccountBalance() on a Bank component. Components such as
these do not need a temporal interaction contract to specify their behavior because their current behavior does not depend on their past.

However, in other kinds of systems, there are components which must respond to signals which are asynchronous stimuli communicated between instances. Also, the current behavior of the component depends on its past behavior. For example, consider a component that acts as an air-to-air missile guidance system. Now, the current behavior of this component will depend on its past behavior. Also, time plays a critical role in the execution of this component. For example, the missile should be launched only when an enemy missile is located, and it must be launched within a critical time limit.

In conclusion, the behaviors of components who must respond to asynchronous stimulus or whose current behavior depends on past behavior and where time plays a critical role in the execution are best specified using a temporal interaction contract.

To illustrate temporal interaction contracts, let’s consider a very simple example: Consider an Electric Bulb component. Figure 3.1 represents the temporal interaction contract of Electric Bulb graphically. The Electric Bulb has four states: Start, On, Off and AutoOff. The three events the Electric Bulb can receive are switchOn, dimLights and switchOff.

Every temporal Interaction Contract has a start state (denoted as START) that indicates the start of component execution, and an end state (denoted as END) that indicates when the execution stops. (Sometimes, the component may automatically make a transition from the START state to next state. In such a scenario, the START state acts as a pseudo-state.)
The Electric Bulb receives the event ‘switchOn’ and changes its state to ‘On’ in response to the event. The direction of the message is indicated in the square braces. Here ‘IN’ means that the component accepts this message and ‘OUT’ means that the component emits this message, which in turn can be an input to some other component. When Electric Bulb is in ‘On’ state, it can receive either the ‘dimLights’ or ‘switchOff’ message. When it receives ‘dimLights’ message, bulb decreases its light intensity and remains in the ‘On’ state. When it receives, ‘switchOff’ message, the bulb terminates its execution and shuts down. However, when the bulb is in the ‘On’ state for 20 minutes, it automatically switches itself off. This is denoted by the timed transition (shown by dotted lines) from the ‘On’ state. An interesting thing to be noted here is that the presence of the temporal transition from ‘On’ state places a constraint on all the other messages that originate from this state. Once the bulb enters the ‘On’ state,
'dimLights' and 'switchOff' can be invoked within 20 minutes. This is because, after 20 minutes, the bulb will always transit to the 'AutoOff' state.

In next sections, we look at each element of a Temporal Interaction Contract in detail.

3.1.1. States

A state can be defined as a condition or situation in the life of a component during which it satisfies some conditions, performs some activity or waits for some event. The state of a component varies over time and is mainly determined the values of internal variables of a component.

A state specification consists of the following elements:

1. **Name** – It is a textual string that distinguishes one state from the other. Each state in temporal interaction contract must have a unique name.
2. **Entry/Exit Action** – These actions are executed on entering and exiting the state respectively.
3. **Internal transitions** – These are the transitions that are handled without causing a change in the state.
4. **Substates** – These are states within a state. They can be executed either concurrently or sequentially.

It is very important to decide about the semantics of a state. In the case of the Electric Bulb component discussed before, any microscopic changes in the atoms of Electric Bulb can be thought to constitute a state. Although it is accurate, it will lead to possibly an infinite number of states; each giving very little information. From the point of view of the user of the Electric Bulb, only the **externally visible** state(s) that make(s) a **semantic difference** to the system are ‘Start’, ‘On’ and ‘End’. So a state of a component is created when it causes externally visible semantic difference that conveys significant information.
3.1.2. Transitions

Transitions show movements between states. A transition can be defined as a relationship between two states indicating that a component in the first state will perform certain actions and will enter the second state when specific conditions are satisfied. When such a state change occurs, the transition is said to have been fired. Until the transition fires, the component remains in the source state; after it fires, the component moves to the target state. Interaction Contracts represent only externally visible transitions of a component. Every transition has the following parameters:

1. **Source State** – This is the state affected by the transition. If a component is in the source state, an outgoing transition will fire when the object receives the trigger event of the transition and the guard condition, if any, is satisfied.
2. **Event** – This is an external occurrence that triggers the transition.
3. **Guard** – It is a Boolean condition that must evaluate to true for the transition to occur.
4. **Action** – It is the activity that should be performed when the transition fires.
5. **Target State** – The component moves to this state after the completion of a transition.

3.1.3. Events

An event can be defined as a specification of a noteworthy occurrence that has a location in time. Events trigger transitions in state machines. Events are normally shown externally on transitions. The following types of events are defined:

1. **Temporally Non-Constrained Event** – It is the simplest type of event. It indicates a request for a specific component method to be invoked.
2. **Temporally Constrained Event** – Temporally constrained events are denoted by the value of the associated time threshold. It specifies a
threshold time after which the event is triggered. It is important to make sure that the time units are specified at a uniform level of granularity.

3.2. XML representation of Temporal Interaction Contracts

Fig 3.1 depicts temporal interaction contracts using a Finite State Machine (FSM) notation. The thesis uses FSM notation to describe temporal interaction contracts of components. However, it should be noted that prototype developed for validation acts on XML representation of temporal interaction contracts. For more details on XML representation of temporal interaction contract, refer to Appendix A of the thesis.

3.3. Guidelines for Writing Interaction Contracts

While writing an interaction contract for a component, essentially three things should be specified: the events to which the component can respond, the response to those events and the impact of the past on the current behavior. It is also important to decide on the order in which the component can meaningfully respond to the events, starting at the time of the beginning of component execution until its end.

Here are a few guidelines for modeling interaction contracts for a component:

- Establish the initial and final states for the component. To guide the rest of the specification, one can possibly specify pre- and post-conditions of the initial and final states, respectively.
- Decide on the events to which this component will possibly respond. Typically, these events will correspond to method invocations specified in the provided and required interfaces of the component. Consider the other components with which this component may interact. Then consider the events these other components may possibly dispatch.
- Starting from the initial state to the final state, lay out the top-level states the component may be in. Connect these states with transitions triggered by appropriate events.
- Expand these top-level states as necessary by using substates.
- Check that all events mentioned in the interaction contract match events expected by the component interface. Similarly, check that all events expected by the interface of the component are handled by the interaction contract.
- Make sure that all the actions mentioned in the interaction contract are supported by methods of the components in the system.
- Trace through the interaction contract to check it against expected sequences of events and their responses. Be especially diligent to look out for unreachable states and states which form a cycle where the component may get stuck. If such states are found, rearrange the interaction contract specification to get rid of such problems.
- After rearranging the interaction contract, check it against expected sequences again to ensure that the component's semantics are not changed.

3.4. Component Specification Matching

As discussed earlier in Chapter 1, the demand for software continues to increase and software systems continue to grow in complexity. Component based software development (CBSD) is an emerging field which aims at developing software systems out of prefabricated Commercial off the shelf (COTS) software components. A software component is a unit of composition which has contracts specified in its interface and explicit context dependencies of the component. Software component repositories enable the developer to reuse existing software components to build larger systems. This reuse significantly
reduces the amount of new code that must be written and saves the developer a significant amount of time.

The challenge in software component reuse is to be able to locate the component in the repository. It should be faster and easier to search for the component than to write it from scratch. So, algorithms that automate and formalize the task of component specification matching are of a great benefit. Components can be matched based on two main criteria: compatibility and replaceability. In the context of component interactions, these criteria can be described as follows:

- **Compatibility Analysis** - This refers to checking if the interaction specification of a client and of a server component are compatible, that is, if interactions can take place between these components.
- **Replaceability Analysis** - This refers to checking if a component ‘A’ can replace another component ‘B’ from a component interaction standpoint, that is, if ‘A’ can support the same interactions that ‘B’ supports.

The following sections discuss various algorithms required to perform compatibility and replaceability analysis based on the component interaction specifications.

3.5. **Data Structure Definitions**

This section describes various data structures used in the algorithms. The data structures along with their definitions are as follows:

- **TIC** which stands for Temporal Interaction Contract, is a tuple. Formally we define \( \text{TIC} \equiv (\text{States}, \text{StartState}, \text{EndStates}, \text{Messages}, \text{Transitions}) \) where:
  1. **States** is a set of component states. A state is defined as a condition or situation during the life of a component during which it
satisfies some conditions, performs some activity or waits for some event.

2. **Start** $\subset$ States is the initial state. Every Temporal Interaction Contract has a start state (denoted as ‘START’) that indicates the start of the component’s execution.

3. **End** $\subset$ States is a set of final states. Every Temporal Interaction Contract has at least one end state (denoted as ‘END’) that indicates when the execution stops.

4. **Messages** is a finite set of messages. Formally, we define ‘Messages’ as Messages $\equiv M_{NT} \cup M_{T}$ where
   - $M_{NT}$ is a finite set of non-temporal invocation constraint messages in a TIC. Non-temporal constraints do not have any fixed time value associated with them.
   - $M_{T}$ is a finite set of temporal invocation constraint messages in a TIC. Temporal constraints have a fixed time value associated with them.

5. **Transitions** $\equiv \{T_i\}$ where $T_i = (S_i, S_j, M_k)$ is a finite set of transitions where $S_i, S_j \subset$ States and $M_k \subset$ Messages. Each transition $(S_i, S_j, M_k)$ consists of a source state $S_i$, a target state $S_j$ and a message $M_k$.

   - **TimeInterval** is a structure which holds information about a time duration and a state. Formally, ‘TimeInterval’ is defined as ([$T_1$, $T_2$], S). Here $T_1$ and $T_2$ represent time values, and have the type `float`. S represents a state, and is of the type `String`. Informally, a TimeInterval of ([$T_1$, $T_2$], S) indicates that component will accept a certain message in the time interval from $T_1$ to $T_2$ and if the message is received within this time interval, the component will change its state to S.

   - **InteractionTimeInterval** is a structure which holds information about a time duration and a combination of states. Formally, ‘InteractionTimeInterval’ is defined as ([$T_1$, $T_2$], (S$_1$, S$_2$)). Here $T_1$ and $T_2$
represent time values, and have type `float'. S₁ and S₂ represent two states, and have the type `String'. Informally, InteractionTimeInterval is associated with two components. A InteractionTimeInterval ([T₁, T₂], (S₁, S₂)) indicates that a component can accept (send) a message from (to) another component in the time interval from T₁ to T₂. If the message is received during this time interval, component accepting (sending) the message changes its state to S₁ and component sending (receiving) the messages changes its state to S₂.

- **TimeIntervalSet** is a set whose elements are of type ‘TimeInterval’.
- **InteractionTimeIntervalSet** is a set whose elements are of type ‘InteractionTimeInterval’.
- **TimeIntervalSet InitialTimeSlice[[ ]]** is an array. The array is of dimensions n × m where ‘n’ is the total number of states in a TIC and ‘m’ is the total number of messages in the TIC. InitialTimeSlice stores initial values of time availability for different messages. These values are used for the calculation of TimeSlice.
- **TimeIntervalSet TimeSlice[[ ]]** is an array. The array is of dimensions n × m where ‘n’ is total number of states in a TIC and ‘m’ is total number of messages in TIC. ‘TimeSlice’ will be used to store the information about all the methods available at a given state and associated time constraints.
- **String StateSet** is a sequence of states. This value represents transitive closure of a state. Transitive closure of a state ‘S’ is defined as states that are reachable from ‘S’ with the help of temporal transitions only.
- **StateSet StateSetArray[ ]** is an array of ‘StateSet’ elements. The array is of dimensions 1 × n where ‘n’ is the total number of states in a TIC.
- **InteractionTimeIntervalSet InteractionTrace[[ ][ ]]** is an array. The array is of dimensions n × n × m where ‘n’ is total number of states in a TIC and ‘m’ is the total number of messages in the TIC. Informally, ‘InteractionTrace’ will be used to store the information about the interaction trace of two TICs.
- **InteractionTimeIntervalSet IntersectionTrace** is an array. The array is of dimensions $n \times n \times m$ where 'n' is total number of states in a TIC and 'm' is total number of messages in the TIC. Informally, 'IntersectionTrace' will be used to store the information of common messages between two TICs.

- **Input Message** is a message that can be accepted by a component in a certain state $S$.

- **Output Message** is a message that can be emitted by a component in a certain state $S$.

The TIC introduces the notion of a temporal transition in the component interaction specification. The data structure TimeSlice is used to store the information about all the methods that can be invoked from a given state along with the time interval in which they are active. It facilitates the determination of compatibility and replaceability analysis in the TIC.

To illustrate the structure of TimeSlice; consider a TIC where, after entering the initial state $S_0$, if the component receives an input message $M_1$ during the interval $[T_1, T_2]$, then the component state will change to $S_1$; whereas if it is invoked during the interval $[T_3, T_4]$ then the component will move to the state $S_2$ of the TIC. So, the TimeSlice for state $S_0$ and message $M_1$ will be: $(([T_1, T_2], S_1), ([T_3, T_4], S_2))$.

TimeSlice$[s, m]$, an element of TimeSlice array, is defined for a particular state ‘$s$’ and a non-temporal message $m \subset M$. If a message 'm' cannot be invoked from a state 's' then TimeSlice$[s, m] = $ null.

### 3.6. Function Definitions

The functions used in the algorithms are as follows:
float getDuration(String S_1, String M_1) is a function that returns a real number that specifies the temporal activation constraints on M_1, which is an output message in the state S_1.

boolean Transition(String S_1, String S_2, String M_1) is a function that denotes that tuple (S_1, S_2, M_1) \( \in \) Transitions. Here S_1 is the source state, S_2 is target state and M_1 is the associated message that causes the state transition from S_1 to S_2.

String Direction(String S_1, String M_1) is a function that will return value 'IN' if M_1 is an input message for state S_1, 'OUT' if M_1 is an output message for state S_1 or NULL otherwise.

String Spawn(String S_1) is a function that returns a string of temporally non-constraint output messages from a state S_1.

TimeSlice Add(TimeSlice T[s,m], float x) adds value `x` to each member element of TimeSlice[s,m]. So, it is formally defined as Add(T[s,m], x) = ([T_i + x, T_j + x], S); \( \forall ([T_i, T_j]; S) \in T[s,m] \).

float Min(float T_i, float T_j) returns the minimum value of two time values. Formally Min(T_i, T_j) = T_i if T_i < T_j else T_j.

float Max(float T_i, float T_j) returns maximum value of two time values. Formally Max(T_i, T_j) = T_i if T_i > T_j else T_j.

boolean Overlap(((T_1, T_2],S_1),([T_3, T_4],S_2)) returns true if the two time durations [T_1, T_2] and [T_3, T_4] overlap. Formally, Overlap(((T_1, T_2],S_1),([T_3, T_4],S_2)) will return true iff T_1 \leq T_3 \leq T_2.

TimeSlice Normalize(TimeSlice T) returns the normalized form of the input TimeSlice. To normalize a TimeSlice, overlapping TimeSlice tuples that are associated with same state are combined. Formally, \( \forall ([T_i, T_j], S_i) \) and \( ([T_i, T_k], S_j) \) \( \in \) T where S_i = S_j and either T_i \leq T_i \leq T_k or T_i \leq T_j \leq T_k are replaced with \(([\text{Min}(T_i, T_i), \text{Max}(T_j, T_k)], S_i)\).
3.7. Matching Algorithms

This section depicts various algorithms used for matching TICs specifications of two components. In the subsequent discussion, following TICs are used as examples.

Figure 3.2 Temporal Interaction Contract T₁ (Graphical Representation)

For TIC T₁:

1. States ≡ {Start, S2, S3, S4, S5, S6, End}
2. Start ≡ {Start}
3. End ≡ {S3, S4, S6, End}
4. Messages ≡ {m₁, m₂, m₃, m₄, m₅, m₆}
5. Transitions ≡ { (Start, S₂, m₁), (S₂, S₃, m₂), (S₂, S₄, m₃), (S₂, S₅, m₄), (S₅, S₆, m₆), (S₅, End, m₅) }
For TIC $T_2$:

1. States $\equiv \{ \text{Start}', \text{S2}', \text{S3}', \text{S4}', \text{S5}', \text{S6}', \text{End}' \}$
2. Start $\equiv \{ \text{Start}' \}$
3. End $\equiv \{ \text{S3}', \text{S4}', \text{S6}', \text{End}' \}$
4. Messages $\equiv \{ \text{m1}', \text{m2}', \text{m3}', \text{m4}', \text{m5}', \text{m6}' \}$
5. Transitions $\equiv \{ (\text{Start}', \text{S2}', \text{m1}'), (\text{S2}', \text{S3}', \text{m2}'), (\text{S2}', \text{S4}', \text{m3}'), (\text{S5}', \text{S6}', \text{m6}'), (\text{S2}', \text{S5}', \text{m4}'), (\text{S5}', \text{End}', \text{m5}') \}$
Figure 3.4 Temporal Interaction Contract T₃ (Graphical Representation)

For TIC T₃:

1. States ≡ {Start, S2, S3, S4, S5, S6, End}
2. Start ≡ {Start}
3. End ≡ {S3, S4, S6, End}
4. Messages ≡ {m₁, m₂, m₃, m₄, m₅, m₆}
5. Transitions ≡ { (Start, S2, m₁), (S2, S3, m₂), (S2, S4, m₃), (S2, S5, m₄), (S5, S6, m₆), (S5, End, m₅) }

The following sections will depict the algorithms used for checking compatibility and replaceability among TICs of components. As mentioned earlier, TICs T₁, T₂ and T₃ will be used as examples to explain the application of these algorithms.
3.7.1. ‘Initialize’ Algorithm

```
input: A Temporal Interaction Contract
       \[ T = (States, Start, End, Messages, Transitions) \]
output: InitialTimeSlice

1.1 Begin
1.2 Duration = +\infty ;
1.3 foreach s \in States do
1.4     foreach m \in Messages do
1.5         InitialTimeSlice [s,m] = \phi
1.6     end
1.7 end
1.8 foreach s \in States do
1.9     if ((\exists s_1 \in States) \land (Transition(s, s_1, m_{time})) \land (m_{time} \in M_T)) then
1.10         Duration = getDuration(s, m_{time}) ;
1.11         foreach m_{spawn} \in Spawn(s) do
1.12             if ((\exists s_{\text{spawn}} \in States) \land (Transition(s, s_{\text{spawn}}, m_{\text{spawn}}))) then
1.13                 InitialTimeSlice [s, m_{\text{spawn}}] = ([0, Duration], s_{\text{spawn}}) ;
1.14             end
1.15         end
1.16     end
1.17 else
1.18     foreach m_{\text{spawn}} \in Spawn(s) do
1.19         foreach (Transition(s, s_{\text{spawn}}, m_{\text{spawn}})) do
1.20             InitialTimeSlice [s, m_{\text{spawn}}] = ([0, Duration], s_{\text{spawn}}) ;
1.21         end
1.22     end
1.23 end
1.24 Duration = +\infty ;
1.25 end
1.26 Return InitialTimeSlice;
```

Algorithm 1: Initialize
The ‘Initialize’ algorithm is used to deduce the value of InitialTimeSlice array. The values of elements of InitialTimeSlice array are based on the Temporal Interaction Contract (TIC) of the component. Each InitialTimeSlice array element is of type \([t_1, t_2], S_1\). InitialTimeSlice array has ‘States’ as row index and ‘Messages’ as column index. So if InitialTimeSlice \([S_1, m_1] = ([t_1, t_2], S_2);\) it means that, if a component is in state \(S_1\) and it receives message \(m_1\) within time interval \(t_1\) to \(t_2\), then component will change its state to \(S_2\). In other words, in state \(S_1\), message \(m_1\) is valid for time duration \(t_1\) to \(t_2\) and during this time period it causes the component to change its state. Thus, InitialTimeSlice array holds the message availability information for all the states of a component.

The algorithm begins by initializing ‘Duration’ value as positive infinity (line 1.2). All the elements of InitialTimeSlice array are set to ‘NULL’ value (lines 1.3 to 1.7). Then for every state of the component, the algorithm checks whether this state has a temporal transition. If there is a temporal transition, then ‘Duration’ is set to the time value associated with the temporal constraint. Then for all non-temporal messages originating from this state (also called as spawn messages of this state), their availability is set as 0 to ‘Duration’ (lines 1.9 to 1.16). If a state does not have any temporal transitions, then availability of its spawn messages is set as 0 to infinity (lines 1.17 to 1.23).

To conclude, the ‘Initialize’ algorithm returns the InitialTimeSlice array. The InitialTimeSlice array is used to produce the TimeSlice array. Thus ‘Initialize’ algorithm is used to extract initial temporal information from the TIC which is further processed to create TimeSlice array. The InitialTimeSlice arrays calculated for TIC \(T_1\), \(T_2\) and \(T_3\) using the ‘Initialize’ algorithm are as shown below.
Table 3.1 InitialTimeSlice for TIC T₁

<table>
<thead>
<tr>
<th></th>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>{([0, ∞], S2)}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S2</td>
<td>NULL</td>
<td>{([0,10], S3)}</td>
<td>{([0,10], S4)}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S3</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S4</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S5</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>{([0,30], S6)}</td>
</tr>
<tr>
<td>End</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Table 3.2 InitialTimeSlice for TIC T₂

<table>
<thead>
<tr>
<th></th>
<th>m1'</th>
<th>m2'</th>
<th>m3'</th>
<th>m4'</th>
<th>m5'</th>
<th>m6'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start'</td>
<td>{([0, ∞], S2')}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S2'</td>
<td>NULL</td>
<td>{([0,20], S3')}</td>
<td>{([0,20], S4')}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S3'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S4'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S5'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>{([0,40'), S6'})</td>
</tr>
<tr>
<td>End'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
3.7.2. ‘Calculate Time Slice’ Algorithm

This algorithm calculates values of the elements in the TimeSlice array. The TimeSlice array is calculated based on the InitialTimeSlice array. Thus, InitialTimeSlice array (produced by ‘Initialize’ algorithm) acts as an input to this algorithm. The elements of the TimeSlice array are similar to the elements of the InitialTimeSlice array and they have the same semantics. Thus TimeSlice array holds the message availability information about all the states of a component. The TimeSlice value is calculated for a particular state, which is an input to the algorithm along with a TIC.

### Table 3.3 InitialTimeSlice for TIC T₃

<table>
<thead>
<tr>
<th></th>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>{([0, ∞], S2)}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S2</td>
<td>NULL</td>
<td>{([0, 20], S3)}</td>
<td>{([0, 20], S4)}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S3</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S4</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S5</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>{([0, 40], S6)}</td>
</tr>
<tr>
<td>End</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
input: A Temporal Interaction Contract
\[ T = (\text{States, Start, End, Messages, Transitions}) \]

output: TimeSlice

2.1 Begin;
2.2 InitialTimeSlice = Initialize(T);
2.3 while \((\exists s_{\text{input}} \in \text{States})\) do
2.4 \quad LastState = s_{\text{input}}, \text{ TimeElapsed} = 0, \text{ Temp} = \{\phi\},
2.5 \quad \text{StateSetArray} [s_{\text{input}}] = \{s_{\text{input}}\};
2.6 \quad \text{foreach} \text{ message} \in M_{NT} \text{ do}
2.7 \quad \quad \text{TimeSlice} [s_{\text{input}}, \text{ message}] = \{\phi\};
2.8 \quad \text{end}
2.9 \quad \text{while} \ (\text{Temp} \neq \text{StateSetArray} [s_{\text{input}}]) \text{ do}
2.10 \quad \quad \text{Temp} = \text{StateSetArray} [s_{\text{input}}];
2.11 \quad \quad \text{if} \ ((\exists s' \in \text{States}) \land (\text{Transition(LastState,s',m)})) \land \ (m \in M_T) \text{ then}
2.12 \quad \quad \quad \text{StateSetArray} [s_{\text{input}}] = \text{StateSetArray} [s_{\text{input}}] \cup \{s'\};
2.13 \quad \quad \quad \text{TimeElapsed} = \text{TimeElapsed} + \text{getDuration(LastState,m)};
2.14 \quad \quad \quad \text{foreach} m' \in \text{Spawn}(s') \text{ do}
2.15 \quad \quad \quad \quad \text{TimeSlice} [s_{\text{input}},m'] = \text{Normalize(TimeSlice} [s_{\text{input}},m']
2.16 \quad \quad \quad \quad \quad \cup \text{Add(InitialTimeSlice} [s',m'],\text{TimeElapsed})\);\end{end}
2.17 \quad \quad \quad \text{LastState} = s' ;
2.18 \quad \quad \text{end}
2.19 \quad \text{else}
2.20 \quad \quad \text{foreach} m' \in \text{Spawn(LastState)} \text{ do}
2.21 \quad \quad \quad \text{TimeSlice} [s_{\text{input}},m'] = \text{Normalize(TimeSlice} [s_{\text{input}},m']
2.22 \quad \quad \quad \quad \quad \cup \text{Add(InitialTimeSlice} [\text{LastState},m'],\text{TimeElapsed})\);\end{end}
2.23 \quad \text{end}
2.24 \quad \text{end}
2.25 \quad \text{States} = \text{States} - s_{\text{input}} ;
2.26 \text{end}
2.27 \text{Return TimeSlice};

\textbf{Algorithm 2: Calculate TimeSlice}
The algorithm starts by calculating InitialTimeSlice value for the given TIC (line 2.2). Then a state, labeled as ‘s_{input}’ is picked from ‘States’ set (line 2.3). For every non-temporal message, the TimeSlice value for s_{input} is set as ‘NULL’ (line 2.7). The algorithm then proceeds by calculating the transitive closure for the input state (line 2.9 to 2.23). Here, the transitive closure is defined as all the states that are reachable from input states via temporal transitions. For each state in this transitive closure, the TimeSlice value is calculated by incrementally updating the value of the TimeElapsed variable (line 2.13). This variable keeps track of the time required to travel from an input state to the current state in the transitive closure. The current value of ‘TimeElapsed’ variable is obtained by adding all the temporal transition values from the input state to the current state.

After processing for ‘s_{input}’ state is complete, it is deleted from the ‘States’ set. The algorithm then proceeds by picking up the next state from the ‘States’ set and performing the transitive-closure operations described before. The algorithm terminates when ‘States’ set becomes empty. Upon termination, it returns the TimeSlice array. The TimeSlice for TIC T_1, T_2 and T_3 are shown below.

<table>
<thead>
<tr>
<th></th>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>{([0, \infty],S2)}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S2</td>
<td>NULL</td>
<td>{([0,10],S3)}</td>
<td>{([0,10],S4)}</td>
<td>NULL</td>
<td>{([10,40],S6)}</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S4</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S5</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>{([0,30],S6)}</td>
</tr>
<tr>
<td>End</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
### Table 3.5 TimeSlice for TIC T₂

<table>
<thead>
<tr>
<th></th>
<th>m1'</th>
<th>m2'</th>
<th>m3'</th>
<th>m4'</th>
<th>m5'</th>
<th>m6'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>{([0, ∞],S2')}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S2'</td>
<td>NULL</td>
<td>{([0,20],S3')}</td>
<td>{([0,20],S4')}</td>
<td>NULL</td>
<td>NULL</td>
<td>{([20,60],S6')}</td>
</tr>
<tr>
<td>S3'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S4'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S5'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>{([0,40],S6')}</td>
</tr>
<tr>
<td>End'</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Table 3.6 TimeSlice for TIC T₃

<table>
<thead>
<tr>
<th></th>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>{([0, ∞],S2)}</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S2</td>
<td>NULL</td>
<td>{([0,20],S3)}</td>
<td>{([0,20],S4)}</td>
<td>NULL</td>
<td>NULL</td>
<td>{([20,60],S6)}</td>
</tr>
<tr>
<td>S3</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S4</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>S5</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>{([0,40],S6)}</td>
</tr>
<tr>
<td>End</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
3.7.3. ‘Calculate Interaction Trace’ Algorithm

```
\textbf{input} : Two Temporal Interaction Contracts

\[ T_1 = (\text{States}_1, \text{Start}_1, \text{End}_1, \text{Messages}_1, \text{Transitions}_1) \text{ and } \]
\[ T_2 = (\text{States}_2, \text{Start}_2, \text{End}_2, \text{Messages}_2, \text{Transitions}_2) \]

\textbf{output}: Interaction Trace

3.1 \textbf{Begin} ;
3.2 \textbf{Combined} = \{(\text{Start}_1, \text{Start}_2)\}, \text{StartSet} = \{\emptyset\} ;
3.3 \text{TimeSlice}_1 = \text{CalculateTimeSlice}(T_1) ;
3.4 \text{TimeSlice}_2 = \text{CalculateTimeSlice}(T_2) ;
3.5 \textbf{while} \exists (temp_1, temp_2) \in \text{Combined} \textbf{do}
3.6 \hspace{1em} \text{StartSet} = \text{StartSet} \cup \{(temp_1, temp_2)\} ;
3.7 \hspace{1em} \text{Combined} = \text{Combined- StartSet} ;
3.8 \hspace{1em} \textbf{foreach} (m \in \text{Spawn}(temp_1) \land m \in \text{Spawn}(temp_2)) \textbf{do}
3.9 \hspace{2em} \textbf{if} \text{Direction}(temp_1, m) \neq \text{Direction}(temp_2, m) \textbf{then}
3.10 \hspace{3em} \textbf{foreach} ([t_1, t_2], s_1) \in \text{TimeSlice}_1 [temp_1, m] \land ([t_3, t_4], s_2) \in \text{TimeSlice}_2 [temp_2, m] \textbf{do}
3.11 \hspace{4em} \textbf{if} \text{Max}(t_1, t_3) \leq \text{Min}(t_2, t_4) \textbf{then}
3.12 \hspace{5em} \text{minimum} = \text{Min}(t_2, t_4), \text{maximum} = \text{Max}(t_1, t_3) ;
3.13 \hspace{5em} \text{InteractionTrace} [temp_1, temp_2, m] = \text{InteractionTrace}
3.14 \hspace{5em} [temp_1, temp_2, m] \cup \{\text{[(minimum, maximum), (s_1, s_2)]}\} ;
3.15 \hspace{5em} \text{Combined} = \text{Combined} \cup \{(s_1, s_2)\}
3.16 \hspace{4em} \textbf{end}
3.17 \hspace{3em} \textbf{end}
3.18 \hspace{1em} \textbf{end}
3.19 \textbf{Return} \text{InteractionTrace} ;
```

\textbf{Algorithm 3:} Calculate Interaction Trace Between Two TICs
This algorithm calculates the interaction trace between two TICs. This interaction trace will contain all possible interactions that can occur between these two components. The interaction trace can be used to determine the partial compatibility between two TICs. The algorithm uses TimeSlice values of both the TICs. These TimeSlice values are calculated by the ‘Calculate TimeSlice’ algorithm described earlier. The interaction trace information is stored in an array named ‘InteractionTrace’. So, if InteractionTrace[s1, s2, m1] = ([t1, t2], (s3, s4)) then it means message m1 is common between states s1 and s2 of TIC1 and TIC2 respectively. This message has opposite directions in TIC1 and TIC2 i.e. it has polarity ‘IN’ in one TIC and ‘OUT’ in the other TIC. Thus this message represents a possible interaction between these two components. Also, this interaction is valid only for time period t1 to t2. If this interaction occurs within the given time period, TIC1 will transit to state s3 and TIC2 will transit to state s4.

The algorithm uses a set named ‘Combined’ to store the state information. The states to be processed are picked up from this set. The elements of this set are of the form (s1, s2) where s1 is a state from TIC1 and s2 is a state from TIC2.

At the beginning of the algorithm, the start states of both TICs are put in the ‘Combined’ set (line 3.2). The algorithm then proceeds by calculating TimeSlice1 and TimeSlice2, which store the Time Slice values for T1 and T2, respectively. Now until the ‘Combined’ set becomes empty, each time, the first state tuple is picked from this set (line 3.5, 3.6). Now the algorithm determines if these states have any messages that can interact with each other (line 3.8, 3.9). Two states (each belonging to a different TIC) are said to interact if they have a message which is common to both and has opposite direction. If such message(s) is found, then the algorithm checks if their TimeSlice values overlap (line 3.10, 3.11). Overlap check is done for the elements of TimeSlice array. Two TimeSlice array elements ([t_i, t_j], s_k) and ([t_m, t_n], s_p) are said to overlap iff Max(t_i, t_m) <= Min(t_j, t_n). If such overlapping value(s) is found, it is put in the
InteractionTrace (line 3.13). Also, the state information in the time slice is added as a tuple at the end of the ‘Combined’ array (line 3.14).

The algorithm terminates when the ‘Combined’ set becomes empty. This indicates that all the required states have been processed and all the possible interactions have been identified.

To conclude, the ‘Calculate Interaction Trace’ algorithm returns InteractionTrace array. The InteractionTrace array calculated for TIC T₁ and TIC T₂ is as shown below.

\[
\text{InteractionTrace}[S2, S2', m2] = \{(0,10), (S3, S3')\}
\]
\[
\text{InteractionTrace}[S2, S2', m3] = \{(0,10), (S4, S4')\}
\]
\[
\text{InteractionTrace}[S2, S2', m6] = \{(20,40), (S6, S6')\}
\]
\[
\text{InteractionTrace}[S5, S5', m6] = \{(0,30), (S6, S6')\}
\]

All other elements of the array InteractionTrace are NULL. If InteractionTrace[Sᵢ, Sⱼ, mₖ] is NULL then, it indicates that state Sᵢ of TIC₁ and Sⱼ of TIC₂ do not interact by exchanging message mₖ.

3.7.4. ‘Calculate Intersection Trace’ Algorithm

This algorithm calculates an intersection trace between two TICs. This intersection trace will contain all possible messages that are common between these two components. The intersection trace can be used to determine the partial replaceability between two TICs. The algorithm accepts TimeSlice values of both the TICs as input. These TimeSlice values are calculated by the ‘Calculate TimeSlice’ algorithm described earlier. The intersection trace information is stored in an array named ‘IntersectionTrace’. So, if IntersectionTrace[s₁, s₂, m₁] = ([t₁, t₂], (s₃, s₄)) then it means message m₁ is common between states s₁ and s₂ of TIC₁ and TIC₂ respectively. This message has same polarity (either ‘IN’ or ‘OUT’) in both TIC₁ and TIC₂. Thus, this message
represents a possible intersection between these two components. Also, this intersection is valid only for time period \( t_1 \) to \( t_2 \). If this message is invoked in the given time period, TIC\(_1\) will transit to state \( s_3 \) and TIC\(_2\) will transit to state \( s_4 \).

\[
\text{input} : \text{Two Temporal Interaction Contracts} \\
T_1 = (\text{States}_1, \text{Start}_1, \text{End}_1, \text{Messages}_1, \text{Transitions}_1) \text{ and} \\
T_2 = (\text{States}_2, \text{Start}_2, \text{End}_2, \text{Messages}_2, \text{Transitions}_2) \\
\text{output: IntersectionTrace}
\]

4.1 Begin;
4.2 Combined = \{\{\text{Start}_1, \text{Start}_2\}\}, StartSet = \{\emptyset\};
4.3 TimeSlice1 = CalculateTimeSlice(\text{T}_1);
4.4 TimeSlice2 = CalculateTimeSlice(\text{T}_2);
4.5 \textbf{while} \ \exists (temp_1, temp_2) \in \text{Combined} \ \textbf{do}
4.6 \ \ \ \text{StartSet} = \text{StartSet} \cup \{(temp_1, temp_2)\};
4.7 \ \ \ \text{Combined} = \text{Combined- StartSet};
4.8 \ \ \ \textbf{foreach} (m \in \text{Spawn}(temp_1) \land m \in \text{Spawn}(temp_2)) \ \textbf{do}
4.9 \ \ \ \ \ \ \ \textbf{if} \ \text{Direction}(temp_1, m) = \text{Direction}(temp_2, m) \ \textbf{then}
4.10 \ \ \ \ \ \ \ \ \textbf{foreach} \ \begin{cases} 
([t_1, t_2], s_1) \in \text{TimeSlice1 [temp_1, m]} \land ((t_3, t_4), s_2) \\
\in \text{TimeSlice2 [temp_2, m]} 
\end{cases}
4.11 \ \ \ \ \ \ \ \ \textbf{if} \ \text{Max}(t_1, t_3) \leq \text{Min}(t_2, t_4) \ \textbf{then}
4.12 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \text{minimum} = \text{Min}(t_2, t_4), \ \text{maximum} = \text{Max}(t_1, t_3) ;
4.13 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \text{[temp}_1, \text{temp}_2, \text{m}] = \text{IntersectionTrace} \ \text{[temp}_1, \text{temp}_2, \text{m}] \\
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \cup \{(\text{minimum}, \text{maximum }], (s_1, s_2))\};
4.14 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \text{Combined} = \text{Combined} \cup \{(s_1, s_2)\}
4.15 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \end
4.16 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \end
4.17 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \end
4.18 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \end
4.19 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \end
4.20 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \end
\textbf{Return IntersectionTrace};

Algorithm 4: Calculate Intersection of two TICs
The algorithm functions in a manner similar to that of “Calculate Interaction Trace” algorithm. The only difference is that the opposite polarity check in ‘Interaction Trace’ algorithm (line 3.9) is replaced with equality check (line 4.9). This is because, for calculating the intersection of two TICs, we need to check both the TICs for messages of the same polarity.

To conclude, the ‘Calculate Intersection Trace’ algorithm returns IntersectionTrace array. The IntersectionTrace array calculated for TIC T₁ and TIC T₃ is as shown below.

IntersectionTrace[S₂, S₂, m₂] = { ( [0,10], (S₃, S₃) ) }  
IntersectionTrace[S₂, S₂, m₃] = { ( [0,10], (S₄, S₄) ) }  
IntersectionTrace[S₂, S₂, m₆] = { ( [20,40], (S₆, S₆) ) }  
IntersectionTrace[S₅, S₅, m₆] = { ( [0,30], (S₆, S₆) ) }  

All other elements of the array IntersectionTrace are NULL. If IntersectionTrace[Sᵢ, Sⱼ, mₖ] is NULL then, it indicates that state Sᵢ of TIC₁ and Sⱼ of TIC₂ do not have message mₖ as a common message.

3.7.5. ‘Contains’ Algorithm

This algorithm checks if time duration [t₁, t₂] is contained in a TimeSlice T[s, m]. The algorithm searches a set of n elements (([t₁₁, t₂₁], sₖ₁), ([t₁₂, t₂₂], sₖ₂), .. , ([tᵢₙ, tᵢₙ], sₖₙ)) ∈ T[s, m] such that the time interval [t₁, t₂] is contained by the union of these n elements. The ‘Contains’ algorithm is used in subsequent algorithms to check total compatibility and replaceability.

The algorithm starts by checking if there is a single TimeSlice element which can cover the time duration [t₁, t₂] (line 5.3, 5.4 and 5.5). If such an element is not found, then algorithm checks if it can find a set of TimeSlice elements which cover the time duration (line 5.6 to 5.9). If such set does not exist
then the algorithm concludes that time duration \([t_1, t_2]\) is not contained by 
\(\text{TimeSlice}[s, m]\) (line 5.10, 5.11 and 5.12).

\[
\text{Algorithm 5: Algorithm to check Contains constraint}
\]

Let’s consider an example. Suppose \([t_i, t_j] = [0, 40]\). Then it is contained by 
the union of elements of the following set: \(S = \{ ([0, 20], S_2), ([18, 30], S_3), 
([30, 45], S_4) \}\). So, if for a certain \(T[s, m]\), if \(S \subseteq T[s, m]\), then \(T[s, m]\) contains \([0, 40]\). Figure 4.4 describes the idea of ‘Contains’ constrains in a pictorial manner.
3.7.6. ‘Equivalent under Compatibility Constraint’ Algorithm

This algorithm checks if a TIC $T_1$ is equivalent to another TIC $T_2$ under compatibility constraint. A TIC, $T_1$, is said to be 'equivalent under compatibility constraint' to another TIC, $T_2$, iff, starting from the initial state, every input message of $T_1$ can be contained by an output message of $T_2$ or every output message of $T_1$ can be contained by an input message of $T_2$. This equivalence is used to determine complete compatibility between two TICs. The algorithm uses TimeSlice values of both the TICs. These TimeSlice values are calculated by the ‘Calculate TimeSlice’ algorithm described earlier.

The algorithm stores the states information in a set named ‘Combined’. The states to be processed are picked up from this set. The elements of this set are of the form $(s_1, s_2)$ where $s_1$ is a state from TIC$_1$ and $s_2$ is a state from TIC$_2$. 

Figure 3.5 ‘Contains’ Constraint
input: Two Temporal Interaction Contracts

$T_1 = (States_1, Start_1, End_1, Messages_1, Transitions_1)$ and
$T_2 = (States_2, Start_2, End_2, Messages_2, Transitions_2)$

output: 'True' if $T_1$ is equivalent to $T_2$ under compatibility constraint else
'False'

6.1 Begin;
6.2 Combined = $\{(Start_1, Start_2)\}$, StartSet = $\{\phi\}$, Equivalent = False;
6.3 TimeSlice1 = CalculateTimeSlice($T_1$);
6.4 TimeSlice2 = CalculateTimeSlice($T_2$);
6.5 while $\exists (temp_1, temp_2) \in$ Combined do
6.6 StartSet = StartSet $\cup \{(temp_1, temp_2)\}$;
6.7 Combined = Combined - StartSet;
6.8 foreach ($m \in$ Spawn($temp_1$) $\land m \in$ Spawn($temp_2$)) do
6.9 if Direction ($temp_1, m$) $\neq$ Direction($temp_2, m$) then
6.10 foreach ($([t_1, t_2], s_1) \in$ TimeSlice1 $[temp_1, m]$) do
6.11 if Contains([t_1, t_2], s_1), TimeSlice2 $[temp_2, m]$ then
6.12 Equivalent = True;
6.13 else
6.14 Return FALSE;
6.15 end
6.16 if ($\exists ([t_3, t_4], s_2) \in$ $[temp_2, m]$) $\land$ ($\text{Max}(t_1, t_3) \leq \text{Min}(t_2, t_4)$)
6.17 then
6.18 Combined = Combined $\cup \{(s_1, s_2)\}$
6.19 end
6.20 end
6.21 end
6.22 end
6.23 end
6.24 Return Equivalent;

Algorithm 6: Check if Two Temporal Interaction Contracts are Equivalent
under compatibility constraint
At the beginning of the algorithm, the start states of both TICs are put in the ‘Combined’ set (line 6.2). The algorithm then proceeds by calculating TimeSlice1 and TimeSlice2. It stores the Time Slice values for T1 and T2, respectively. Now until the ‘Combined’ set becomes empty, each time, first state tuple is picked from this set (line 6.5, 6.6, 6.7), and the algorithm determines if these states have any messages that can interact with each other (line 6.8, 6.9). If such message(s) is found, then the algorithm checks if each element of TimeSlice1 is contained by TimeSlice2 (line 6.10, 6.11). If the contains constraint is satisfied, then the algorithm continues, else the algorithm terminates and returns FALSE as result (line 6.15). Also, if an element of TimeSlice2 overlaps with the current element from TimeSlice1, then the ‘Combined’ set is updated accordingly. (line 6.18).

3.7.7. ‘Equivalent under Replaceability Constraint’ Algorithm

This algorithm checks if a TIC T1 is equivalent to another TIC T2 under replaceability constraint. A TIC, T1, is said to be ‘equivalent under replaceability constraint’ to another TIC, T2, iff, starting from the initial state, every input message of T1 can be contained by an input message of T2 or every output message of T1 can be contained by an output message of T2. This equivalence is used to determine complete replaceability between two TICs. The algorithm uses TimeSlice values of both the TICs. These TimeSlice values are calculated by the ‘Calculate TimeSlice’ algorithm described earlier.

The algorithm functions in a manner similar to that of ‘Equivalent under Compatibility Constraint’ algorithm. The only difference is that the opposite polarity check in ‘Equivalent under Compatibility Constraint’ algorithm (line 6.9) is replaced with equality check (line 7.9). This is because, for checking replaceability of two TICs, we need to check both the TICs for messages of same polarity.
input: Two Temporal Interaction Contracts

\[ T_1 = (States_1, Start_1, End_1, Messages_1, Transitions_1) \] and
\[ T_2 = (States_2, Start_2, End_2, Messages_2, Transitions_2) \]

output: 'True' if \( T_1 \) is equivalent to \( T_2 \) under replaceability constraint else 'False'

1. Begin;
2. \( \text{Combined} = \{(Start_1, Start_2)\}, \text{StartSet} = \{\phi\}, \text{Equivalent} = \text{False} ; \)
3. \( \text{TimeSlice1} = \text{CalculateTimeSlice}(T_1) ; \)
4. \( \text{TimeSlice2} = \text{CalculateTimeSlice}(T_2) ; \)
5. \( \text{while } \exists (\text{temp}_1, \text{temp}_2) \in \text{Combined} \text{ do} \)
6. \( \quad \text{StartSet} = \text{StartSet} \cup \{\langle \text{temp}_1, \text{temp}_2 \rangle \} ; \)
7. \( \quad \text{Combined} = \text{Combined} - \text{StartSet} ; \)
8. \( \quad \text{foreach } (m \in \text{Spawn(\text{temp}_1)} \land m \in \text{Spawn(\text{temp}_2)}) \text{ do} \)
9. \( \quad \quad \text{if Direction } (\text{temp}_1, m) = \text{Direction}(\text{temp}_2, m) \text{ then} \)
10. \( \quad \quad \quad \text{foreach } ([t_1, t_2], s_1) \in \text{TimeSlice1} [\text{temp}_1, m] \text{ do} \)
11. \( \quad \quad \quad \quad \text{if Contains}([t_1, t_2], s_1), \text{TimeSlice2} [\text{temp}_2, m] \text{ then} \)
12. \( \quad \quad \quad \quad \quad \text{Equivalent} = \text{True} ; \)
13. \( \quad \quad \quad \text{else} \)
14. \( \quad \quad \quad \quad \quad \text{Return FALSE} ; \)
15. \( \quad \quad \text{end} \)
16. \( \quad \text{end} \)
17. \( \quad \text{if } (\exists ([t_3, t_4], s_2) \in [\text{temp}_2, m]) \land (\text{Max}(t_1, t_3) \leq \text{Min}(t_2, t_4)) \text{ then} \)
18. \( \quad \quad \text{Combined} = \text{Combined} \cup \{\langle s_1, s_2 \rangle\} \)
19. \( \quad \text{end} \)
20. \( \text{end} \)
21. \( \text{end} \)
22. \( \text{end} \)
23. \( \text{Return Equivalent;} \)

\textbf{Algorithm 7}: Check if Two Temporal Interaction Contracts are Equivalent under replaceability constraint
3.7.8. ‘Check Partial Compatibility’ Algorithm

This algorithm determines if two TICs, $T_1$ and $T_2$, are partially compatible. $T_1$ is said to be partially compatible with $T_2$ iff its Interaction Trace with $T_2$ is not null.

\[
\begin{align*}
\text{input} & : \text{Two Temporal Interaction Contracts} \\
T_1 & = (\text{States}_1, \text{Start}_1, \text{End}_1, \text{Messages}_1, \text{Transitions}_1) \text{ and} \\
T_2 & = (\text{States}_2, \text{Start}_2, \text{End}_2, \text{Messages}_2, \text{Transitions}_2) \\
\text{output} & : \text{‘YES’ if } T_1 \text{ is partially compatible with } T_2 \text{ else ’NO’}
\end{align*}
\]

8.1 Begin ;

8.2 $\text{InteractionTrace} = \text{CalculateInteractionTrace}(T_1, T_2) ;$

8.3 if ($\text{InteractionTrace} = \text{NULL}$) then

8.4 Return NO

8.5 end

8.6 else

8.7 Return YES

8.8 end

Algorithm 8: Algorithm to Check Partial Compatibility
3.7.9. ‘Check Complete Compatibility’ Algorithm

This algorithm determines if two TICs, $T_1$ and $T_2$, are completely compatible. $T_1$ is said to be completely compatible with $T_2$ iff $T_1$ is equivalent to $T_2$ under the compatibility constraint.

Algorithm 10: Algorithm to Check Complete Compatibility

```
input : Two Temporal Interaction Contracts

$T_1 = (States_1, Start_1, End_1, Messages_1, Transitions_1)$ and
$T_2 = (States_2, Start_2, End_2, Messages_2, Transitions_2)$

output: ‘TRUE’ if $T_1$ is completely compatible with $T_2$ else ‘FALSE’

10.1 Begin ;
10.2 Equivalent = EquivalentUnderCompatibilityConstraint($T_1, T_2$) ;
10.3 if (Equivalent = FALSE) then
10.4     Return NO
10.5 end
10.6 else
10.7     Return YES
10.8 end
```
3.7.10. ‘Check Partial Replaceability’ Algorithm

This algorithm determines if two TICs, $T_1$ and $T_2$, are partially replaceable. $T_1$ is said to be partially replaceable with $T_2$ iff its Intersection Trace with $T_2$ is not null.

\begin{algorithm}
\textbf{input} : Two Temporal Interaction Contracts

\begin{align*}
T_1 &= (States_1, \text{Start}_1, \text{End}_1, Messages_1, Transitions_1) \text{ and} \\
T_2 &= (States_2, \text{Start}_2, \text{End}_2, Messages_2, Transitions_2)
\end{align*}

\textbf{output}: 'TRUE' if $T_1$ is partially replaceable with $T_2$ else 'FALSE'

9.1 Begin \\
9.2 IntersectionTrace = CalculateIntersection($T_1$, $T_2$) ; \\
9.3 if (IntersectionTrace = NULL) then \\
9.4 \quad Return NO \\
9.5 end \\
9.6 else \\
9.7 \quad Return YES \\
9.8 end
\end{algorithm}

Algorithm 9: Algorithm to Check Partial Replaceability
3.7.11. ‘Check Complete Replaceability’ Algorithm

This algorithm determines if two TICs, \( T_1 \) and \( T_2 \), are completely replaceable. \( T_1 \) is said to be completely replaceable with \( T_2 \) iff \( T_1 \) is equivalent to \( T_2 \) under the replaceability constraint.

| input | Two Temporal Interaction Contracts
|---|---|
| \( T_1 = (States_1, Start_1, End_1, Messages_1, Transitions_1) \) | \( T_2 = (States_2, Start_2, End_2, Messages_2, Transitions_2) \)

output: ‘TRUE’ if \( T_1 \) is completely replaceable with \( T_2 \) else ‘FALSE’

11.1 Begin;
11.2 Equivalent = EquivalentUnderReplaceabilityConstraint(\( T_1, T_2 \));
11.3 if (Equivalent = FALSE) then
11.4 Return NO
11.5 end
11.6 else
11.7 Return YES
11.8 end

Algorithm 11: Algorithm to Check Complete Compatibility

3.8. Control Flow for Determining Partial Compatibility

Fig. 3.6 depicts the sequence in which the proposed algorithms are used to determine if two TICs are partially compatible. Let’s assume that these two TICs are named \( T_1 \) and \( T_2 \), respectively. First, InitialTimeSlice values for both TICs (InitialTimeSlice1 and InitialTimeSlice2) are calculated. Then these values are used to calculate TimeSlice values of both TICs (TimeSlice1 and TimeSlice2). Then TimeSlice1 and TimeSlice2 are used to calculate InteractionTrace of \( T_1 \) and \( T_2 \). Then based on this InteractionTrace, ‘Check Partial Compatibility’ algorithm determines if \( T_1 \) and \( T_2 \) are partially compatible.
Figure 3.6 Determine Partial Compatibility
3.9. Control Flow for Determining Partial Replaceability

Fig. 3.7 depicts the sequence in which the proposed algorithms are used to determine if two TICs are partially replaceable. Let’s assume that these two TICs are named T\textsubscript{1} and T\textsubscript{2}, respectively. First, InitialTimeSlice values for both TICs (InitialTimeSlice\textsubscript{1} and InitialTimeSlice\textsubscript{2}) are calculated. Then these values are used to calculate TimeSlice values of both TICs (TimeSlice\textsubscript{1} and TimeSlice\textsubscript{2}). Then TimeSlice\textsubscript{1} and TimeSlice\textsubscript{2} are used to calculate IntersectionTrace of T\textsubscript{1} and T\textsubscript{2}. Then based on this IntersectionTrace, ‘Check Partial Replaceability’ algorithm determines if T\textsubscript{1} and T\textsubscript{2} are partially replaceable.

3.10. Control Flow for Determining Complete Compatibility

Fig. 3.8 depicts the sequence in which the proposed algorithms are used to determine if two TICs are completely compatible. Let’s assume that these two TICs are named T\textsubscript{1} and T\textsubscript{2}, respectively. First, InitialTimeSlice values for both TICs (InitialTimeSlice\textsubscript{1} and InitialTimeSlice\textsubscript{2}) are calculated. Then these values are used to calculate TimeSlice values of both TICs (TimeSlice\textsubscript{1} and TimeSlice\textsubscript{2}). Then TimeSlice\textsubscript{1} and TimeSlice\textsubscript{2} are used by the ‘Equivalence under Compatibility Constraint’ algorithm to determine if T\textsubscript{1} is equivalent to T\textsubscript{2}. This equivalence checking algorithm uses the ‘Contains’ algorithm internally to determine the equivalence. Then based on the result of the equivalence check, the ‘Check Complete Compatibility’ algorithm determines if T\textsubscript{1} and T\textsubscript{2} are completely compatible.
Figure 3.7 Determine Partial Replaceability
Figure 3.8 Determine Complete Compatibility
Figure 3.9 Determine Complete Replaceability

1. **TIC₁**
   - Initialize
     - **InitialTimeSlice₁**
   - Calculate TimeSlice
     - **TimeSlice₁**
   - **Equivalent under Replaceability Constraint**
     - Check Contains Constraint
       - TRUE / FALSE
   - Check Complete Replaceability
     - YES / NO

2. **TIC₂**
   - Initialize
     - **InitialTimeSlice₂**
   - Calculate TimeSlice
     - **TimeSlice₂**
   - **Equivalent under Replaceability Constraint**
     - Check Contains Constraint
       - TRUE / FALSE
   - Check Complete Replaceability
     - YES / NO
3.11. Control Flow for Determining Complete Replaceability

Fig. 3.9 depicts the sequence in which the proposed algorithms are used to determine if two TICs are completely replaceable. Let’s assume that these two TICs are named $T_1$ and $T_2$, respectively. First, InitialTimeSlice values for both TICs (InitialTimeSlice1 and InitialTimeSlice2) are calculated. Then these values are used to calculate TimeSlice values of both TICs (TimeSlice1 and TimeSlice2). Then TimeSlice1 and TimeSlice2 are used by the ‘Equivalence under Replaceability Constraint’ algorithm to determine if $T_1$ is equivalent to $T_2$. This equivalence checking algorithm uses the ‘Contains’ algorithm internally to determine the equivalence. Then based on the result of the equivalence check, the ‘Check Complete Replaceability’ algorithm determines if $T_1$ and $T_2$ are completely replaceable.

3.12. Conclusion

In this chapter, component compatibility and replaceability algorithms were discussed in the context of temporal interaction contracts. The next chapter will discuss the case study used to validate the algorithms developed in this chapter.
CHAPTER 4. CASE STUDY

The previous chapter discussed the specification of a software component's temporal interaction contract and matching the temporal interaction contracts to determine the compatibility and replaceability of software components. This chapter provides two case studies to illustrate the creation of the temporal interaction contracts and matching them. Two examples are considered for case study: Telephone System and E-commerce system. Section 4.2 describes the Telephone System in detail and the section 4.3 describes the e-commerce system in detail.

4.1. TIC Prototype (TICP) and Experimental Setup

A prototype is developed (named TIC Prototype or TICP), which implements all the TIC matching algorithms discussed in Chapter 3. The prototype is implemented using the Java 2 Platform, Standard Edition (J2SE) version 1.4.2 software environment. The core entities in TICP are ‘Temporal Interaction Contract’, ‘Time Slice’, ‘Interaction Trace’ and ‘Intersection Trace’. These entities are implemented as Java Beans. TICP also contains the program to simulate the behavior specified by a TIC. These simulations can be used to validate the compatibility and replaceability results obtained from the matching algorithms. The TIC matching algorithms use these bean components to perform further analysis. The hardware platform used for the experimentation purpose is the Sun Solaris Ultra-250 Sparc machines, hosting Sun OS release 5.8. The XML representation of all the TICs discussed in this chapter is available in Appendix A. The source code of the prototype is available in Appendix B.
4.2. Telephone System

Consider a Telephone system shown in Fig. 4.1. The Telephone system consists of three components. First is a ‘Dialer’ component which dials the number. The ‘Dialer’ component interacts with the ‘Telephone’ component. The ‘Telephone’ component interacts with the ‘Telephone Exchange’ component. The task of the ‘Telephone’ component is to accept a number from the ‘Dialer’ component and call that number using the functionality provided by the ‘Telephone Exchange’ component. The ‘Telephone’ and ‘Dialer’ components together form a server-client pair where the ‘Telephone’ component acts as server and ‘Dialer’ component acts as client.

The ‘Telephone’ component implements various methods which can be invoked by the ‘Dialer’ component to make a call. These methods are shown in the ‘Telephone’ component in Fig 4.1. Since the ‘Dialer’ acts as the client to the ‘Telephone’ component, it does not support any methods which can be invoked by the ‘Telephone’ component.

The following section describes the temporal interaction contracts for the ‘Telephone’ and ‘Dialer’ components. It is followed by a discussion on the compatibility of these two components. The reasons for compatibility are mentioned and the results obtained from the prototype are shown to substantiate the compatibility claim. The discussion also includes a replaceability analysis of two ‘Telephone’ component TICs. Again, the reasons for compatibility are mentioned and the results obtained from the prototype are shown to substantiate the compatibility claim.
4.2.1. Temporal Interaction Contract for ‘Telephone’ component

The graphical representation of the temporal interaction contract of the ‘Telephone’ component is shown in Fig 4.2. It is assumed that when a component receives a message, ‘m’, it invokes method ‘m()’ of the component. In other words, each message reception corresponds to a method invocation. The ‘Telephone’ component starts in a state called ‘Start’. When the telephone receiver is placed on the hook, ‘Telephone’ receives an ‘onHook’ message from ‘Dialer’ (as explained earlier, it will invoke the ‘onHook()’ method). The Telephone component now enters the ‘On Hook’ state.
Figure 4.2 TIC of ‘Telephone’ Component (Graphical Representation)
When the user picks up the receiver, the ‘Dialer’ component sends an ‘offHook’ message to the ‘Telephone’ component. This causes ‘Telephone’ component to transit to the ‘Off Hook’ state. Now, if user does not dial any number for 10 seconds, the ‘Telephone’ component automatically transits to ‘Put On Hook’ state. This is the end state, which forces the user to put receiver back on the hook and try dialing the number again. When the user starts dialing, the ‘Dialer’ component sends ‘startDialing’ message, which causes the ‘Telephone’ component to transit to the ‘Dialing’ state. The ‘Telephone’ component remains in this state as long as the dialing continues. When user is done dialing, the ‘Dialer’ component sends the ‘doneDialing’ message to the ‘Telephone’ component. This causes the ‘Telephone’ component to transit to the ‘Route Call’ state.

When the ‘Telephone’ component is in the ‘Route Call’ state, it makes the connection with the help of the ‘Telephone Exchange’ component, and the user can start talking. But the user must pay money for the call. If user does not put in any money for 20 seconds, then the ‘Telephone’ component automatically transits to the ‘No Money’ state. The call gets disconnected in this state. However, if the user deposits money, then the ‘Dialer’ sends the ‘moneyDesposited’ message to the ‘Telephone’ component. This message causes the ‘Telephone’ component to change its state to the ‘Money Deposited’.

However, if the line is busy i.e., if there is no response from the other end for 20 seconds, then the ‘Telephone’ component automatically transits to the ‘Return Money’ state. This causes the call to be disconnected and the money is returned back to the user. However, if the ‘Telephone’ component receives the ‘talk’ message in the ‘Money Deposited’ state, then it transits to the ‘CanTalk’ state. When it receives the ‘doneTalking’ message (indicating the user has finished the conversation), it transits to the ‘DoneTalking’ state and stops the execution.
4.2.2. Temporal Interaction Contract for ‘Dialer’ component

Figure 4.3 TIC of ‘Dialer’ Component (Graphical Representation)
The graphical representation of the temporal interaction contract of the ‘Dialer’ component is shown in Fig. 4.3. The ‘Dialer’ component interacts with the user of the telephone system and passes the actions performed by user to the ‘Telephone’ component. The ‘Dialer’ component accomplishes this task by invoking various methods provided by the ‘Telephone’ component. The ‘Dialer’ component sends appropriate messages to the ‘Telephone’ component which causes ‘Telephone’ component to invoke the corresponding methods. As explained earlier, upon receipt of message ‘m’, the component invokes method m().

The ‘Dialer’ component starts in the ‘Receiver on Hook’ state. It sends the ‘onHook’ message and transits to the ‘Receiver picked up’ state. Now when the user lifts the receiver, the ‘Dialer’ component sends the ‘offHook’ message and changes its state to the ‘Accept Dialed Numbers’. When the user starts dialing, the ‘Dialer’ component sends the ‘startDialing’ message and enters the ‘Dialing Done’ state. However, if the user does not dial for 15 seconds, the ‘Dialer’ component automatically transits to the ‘End’ state which stops its execution. It remains in this state as long as the user is dialing. When the user is done dialing, the ‘Dialer’ component sends the ‘doneDialing’ message and changes its state to the ‘Accept Money’ state. When the user deposits money in the coin slot, the ‘Dialer’ component sends the ‘moneyDeposited’ message and changes its state to the ‘Money in Slot’. However, if the user does not deposit money within 10 seconds, the ‘Dialer’ component automatically changes its state to ‘Slot Empty’ and terminates.

After putting money in the slot, the user can start talking. This causes the ‘Dialer’ component to send the message ‘talk’ to the ‘Telephone’ component and the ‘Dialer’ enters the ‘Talking’ state. After the user is done talking, the ‘Dialer’ sends the ‘doneTalking’ message to the ‘Telephone’ component and terminates. While the ‘Dialer’ component is in the ‘Money in Slot’ or ‘Talking’ state, when the
talking time expires, the state automatically changes its state to ‘Time Up’ and terminates execution.

4.2.3. Compatibility Analysis

As described in section 4.2.1 and 4.2.2, the temporal interaction contracts for both the ‘Telephone’ and ‘Dialer’ component contain temporal transitions.

The ‘Telephone’ component has temporal transitions from the ‘Off Hook’ state, ‘Route Call’ state and ‘Money Deposited’ state. These transitions place implicit constraints on the ‘startDialing()’, ‘moneyDeposited()’ and ‘talk()’ methods respectively. The ‘Telephone’ component requires that the ‘startDialing()’ method should be invoked in less than 10 seconds after the component enters the ‘Off Hook’ state. If the method is invoked after 10 seconds, since the ‘Telephone’ component would have moved to ‘Put On Hook’ state, the method invocation is not feasible, i.e., it has no effect on the component. Similarly, the method ‘moneyDeposited()’ should be invoked within 20 seconds once the ‘Telephone’ component enters ‘Route Call’ state otherwise it is not feasible. Also, the ‘talk()’ method should be invoked within 20 seconds once the ‘Telephone’ component enters the ‘Money Deposited’ state; otherwise it can not be invoked.

Similarly, the ‘Dialer’ component has temporal transitions from the ‘Accept Dialed Numbers’, ‘Accept Money’, ‘Money In Slot’ and ‘Talking’ states. These transitions place implicit constraints on the ‘startDialing()’, ‘moneyDeposited()’, ‘talk()’ and ‘doneTalking()’ method calls. When the ‘Dialer’ component enters ‘Accept Dialed Numbers’ state, it can send the ‘startDialing’ message to the ‘Telephone’ component (which causes the ‘Telephone’ component to invoke the ‘startDialing()’ method). But if the ‘Dialer’ component does not send this message for 15 seconds, it automatically enters the ‘End’ state in which it can no longer send the ‘startDialing’ message. Similarly, when the ‘Dialer’ component enters the ‘Accept Money’ state, it can send the ‘moneyDeposited’ message before 10
seconds are over. When the ‘Dialer’ component enters the ‘Money In Slot’ state, it can send the ‘talk’ message for 180 seconds. Similarly, the ‘doneTalking’ message is applicable from the ‘Talking’ state only for 180 seconds after the ‘Dialer’ component enters the ‘Talking’ state.

The computation of the TimeSlice array for the ‘Telephone’ component is as follows. Since the ‘Telephone’ component has eleven states and ten messages, the TimeSlice for ‘Telephone’ component has dimensions of $11 \times 10$. The TimeSlice array for ‘Telephone’ component is as shown below.

\[
\begin{align*}
\text{TimeSlice}[\text{Start, onHook}] &= [0.0, \text{Infinity}] \text{OnHook} \\
\text{TimeSlice}[\text{OnHook, offHook}] &= [0.0, \text{Infinity}] \text{OffHook} \\
\text{TimeSlice}[\text{OffHook, startDialing}] &= [0.0, 10.0] \text{Dialing} \\
\text{TimeSlice}[\text{Dialing, doneDialing}] &= [0.0, \text{Infinity}] \text{RouteCall} \\
\text{TimeSlice}[\text{RouteCall, moneyDeposited}] &= [0.0, 20.0] \text{MoneyDeposited} \\
\text{TimeSlice}[\text{MoneyDeposited, talk}] &= [0.0, 20.0] \text{CanTalk} \\
\text{TimeSlice}[\text{CanTalk, doneTalking}] &= [0.0, \text{Infinity}] \text{DoneTalking} \\
\text{All other elements are NULL.}
\end{align*}
\]

Since ‘Dialer’ component has eleven states and ten messages, its TimeSlice for ‘Dialer’ component has dimensions of $11 \times 10$. The TimeSlice array for ‘Dialer’ component is as shown below.

\[
\begin{align*}
\text{TimeSlice}[\text{ReceiverOnHook, onHook}] &= [0.0, \text{Infinity}] \text{ReceiverPickedUp} \\
\text{TimeSlice}[\text{ReceiverPickedUp, offHook}] &= [0.0, \text{Infinity}] \text{AcceptDialedNumbers} \\
\text{TimeSlice}[\text{AcceptDialedNumbers, startDialing}] &= [0.0, 15.0] \text{DialingDone} \\
\text{TimeSlice}[\text{DialingDone, doneDialing}] &= [0.0, \text{Infinity}] \text{AcceptMoney} \\
\text{TimeSlice}[\text{AcceptMoney, moneyDeposited}] &= [0.0, 10.0] \text{MoneyInSlot} \\
\text{TimeSlice}[\text{MoneyInSlot, talk}] &= [0.0, 180.0] \text{Talking} \\
\text{TimeSlice}[\text{Talking, doneTalking}] &= [0.0, 180.0] \text{Done} \\
\text{All other elements are NULL.}
\end{align*}
\]

These two TICs were fed to the TICP for the partial compatibility check. The output of this analysis is as shown below.
PARTIAL COMPATIBILITY ANALYSIS
==================================

Since InteractionTrace of these two TICs is not null, these two TICs ARE PARTIALLY COMPATIBLE
The non-null elements of the InteractionTrace are as follows:

InteractionTrace[Start, ReceiverOnHook, onHook] = [0.0,Infinity]$OnHook$ReceiverPickedUp
InteractionTrace[OnHook, ReceiverPickedUp, offHook] = [0.0,Infinity]$OffHook$AcceptDialenedNumbers
InteractionTrace[OffHook, AcceptDialenedNumbers, startDialing] = [0.0,10.0]$Dialing$DialingDone
InteractionTrace[Dialing, DialingDone, startDialing] = [0.0,Infinity]$RouteCall$AcceptMoney
InteractionTrace[RouteCall, AcceptMoney, doneDialing] = [0.0,10.0]$MoneyDeposited$MoneyInSlot
InteractionTrace[MoneyDeposited, MoneyInSlot, moneyDeposited] = [0.0,20.0]$CanTalk$Talking
InteractionTrace[CanTalk, Talking, talk] = [0.0,180.0]$DoneTalking$Done

The partial compatibility analysis indicates that these two components are partially compatible, i.e., there are some interactions that can occur between them. These interactions are listed in the output shown above (essentially the non-null elements of InteractionTrace). This interaction trace indicates which messages can be exchanged between the 'Telephone' and 'Dialer' and what is the time frame in which these messages can be exchanged. For example, consider the interaction trace element “InteractionTrace[MoneyDeposited, MoneyInSlot, moneyDeposited] = [0.0,20.0]$CanTalk$Talking”. It indicates that the state 'MoneyDeposited' from the 'Telephone' component and the state 'MoneyInSlot' from the 'Dialer' component can exchange the message 'moneyDeposited' in the time slice of [0, 20], i.e., 20 seconds after the component enter the 'MoneyDeposited' and 'MoneyInSlot' state. If the 'moneyDeposited' message is exchanged in this time duration, then the 'Telephone' component will move to the 'Can Talk' state and the 'Dialer' component will move to the 'Talking' state. So to conclude, the 'Telephone' and
‘Dialer’ components are partially compatible for the given TIC specification. These two TICs were fed to the TICP for the complete compatibility check. The output of this analysis is as shown below.

```
COMPLETE COMPATIBILITY ANALYSIS
==================================
Analysis for state "Start" from first TIC and "ReceiverOnHook" from second TIC
For message "onHook" -
[0.0,Infinity]$OnHook contains [0.0,Infinity]$ReceiverPickedUp

Analysis for state "OnHook" from first TIC and "ReceiverPickedUp" from second TIC
For message "offHook" -
[0.0,Infinity]$OffHook contains [0.0,Infinity]$AcceptDialedNumbers

Analysis for state "OffHook" from first TIC and "AcceptDialedNumbers" from second TIC
For message "startDialing" -
--------------------------------------------------
ERROR!!!!! [0.0,10.0]$Dialing does not contain [0.0,15.0]$DialingDone
--------------------------------------------------

Analysis for state "Dialing" from first TIC and "DialingDone" from second TIC
For message "doneDialing" -
[0.0,Infinity]$RouteCall contains [0.0,Infinity]$AcceptMoney

Analysis for state "RouteCall" from first TIC and "AcceptMoney" from second TIC
For message "moneyDeposited" -
[0.0,20.0]$MoneyDeposited contains [0.0,10.0]$MoneyInSlot

Analysis for state "MoneyDeposited" from first TIC and "MoneyInSlot" from second TIC
For message "talk" -
--------------------------------------------------
ERROR!!!!! [0.0,20.0]$CanTalk does not contain [0.0,180.0]$Talking
--------------------------------------------------

Analysis for state "CanTalk" from first TIC and "Talking" from second TIC
For message "doneTalking" -
[0.0,Infinity]$DoneTalking contains [0.0,180.0]$Done

Analysis for state "DoneTalking" from first TIC and "Done" from second TIC

Since errors were discovered during analysis, these two TICs ARE NOT COMPLETELY COMPATIBLE
Please fix the indicated errors to make these two TICs completely compatible with each other.
```

```
The complete compatibility analysis suggests that the ‘Telephone’ and ‘Dialer’ components are not completely compatible. This is indicated by the errors discovered during the analysis. After correcting those errors, these two TICs can be made completely compatible. For example, consider the error message “ERROR!!!!! [0.0,10.0]$Dialing does not contain [0.0,15.0]$DialingDone”. As indicated in the output, this error occurs for the ‘Off Hook’ state from the ‘Telephone’ component and the ‘Accept Dialed Numbers’ from the ‘Dialer’ component. It indicates that the time slice [0,10] required by the ‘Telephone’ component does not contain time slice [0, 15] offered by the ‘Dialer’ component. So there is a mismatch in timing constraints. In other words, the ‘Telephone’ can accept the ‘startDialogue’ message for 10 seconds when it enters ‘OffHook’ state. But on the other hand, the ‘Dialer’ component can send the ‘startDialing’ message for 15 seconds when it enters the ‘AcceptDialedNumbers’ state. So, there is a time period from 10 seconds to 15 seconds when the ‘Dialer’ component can send the ‘startDialing’ message but the ‘Telephone’ component will not be able to receive it. This is because, according to the TIC of the ‘Telephone’ component, after 10 seconds have elapsed in ‘OffHook’ state, it will automatically transit to the ‘Put On Hook’ state and will be unable to receive this message. To rectify this time mismatch, one possible solution is to reduce the [0, 15] time slice offered by the ‘Dialer’ so that [0, 10] can contain it. So any value less that or equal to ten and greater that zero will satisfy this criteria. So if the [0,15] time slice is changed to a [0, 8] time slice, then the [0, 10] time slice will contain it. It should be noted that exchanging old [0, 15] time slice with the [0, 8] time slice means that in the ‘Dialer’ component, the ‘No Key Pressed’ transition will have a value 8 instead of 10.

Also consider another error indicated in the analysis: “ERROR!!!!! [0.0,20.0]$CanTalk does not contain [0.0,180.0]$Talking”. As discussed earlier, one possible solution for rectifying this time mismatch is to increase the time slice [0, 20] so that it can contain [0, 180]. By increasing the
time slice to [0, 200] from [0, 20], this condition can be met. It should be noted that this change requires changing the value of the ‘No Response’ message in the ‘Telephone’ component. The old value of 20 seconds should be replaced with the new value of 200 seconds. The new ‘Telephone’ and ‘Dialer’ TICs with these two modifications are shown in Fig. 4.4 and 4.5. Note that the change in the TIC is shown in bold font.

4.2.4. Modified TIC and Recomputation of Compatibility

The partial and complete compatibility analysis after these changes is shown in the following discussion. It should be noted that, as expected, these two modified TICs are now partially as well as completely compatible with each other.

```
PARTIAL COMPATIBILITY ANALYSIS
================================
Since InteractionTrace of these two TICs is not null, these two TICs ARE PARTIALLY COMPATIBLE
The non-null elements of the InteractionTrace are as follows:
InteractionTrace[Start, ReceiverOnHook, onHook] = [0.0,Infinity]$OnHook$ReceiverPickedUp
InteractionTrace[OnHook, ReceiverPickedUp, offHook] = [0.0,Infinity]$OffHook$AcceptDialedNumbers
InteractionTrace[OffHook, AcceptDialedNumbers, startDialing] = [0.0,8.0]$Dialing$DialingDone
InteractionTrace[Dialing, DialingDone, startDialing] = [0.0,Infinity]$RouteCall$AcceptMoney
InteractionTrace[RouteCall, AcceptMoney, doneDialing] = [0.0,10.0]$MoneyDeposited$MoneyInSlot
InteractionTrace[MoneyDeposited, MoneyInSlot, moneyDeposited] = [0.0,180.0]$CanTalk$Talking
InteractionTrace[CanTalk, Talking, talk] = [0.0,180.0]$DoneTalking$Done
```
Figure 4.4 Modified TIC of ‘Telephone’ Component (Graphical Representation)
Figure 4.5 Modified TIC of 'Dialer' Component (Graphical Representation)
COMPLETE COMPATIBILITY ANALYSIS
=================================

Analysis for state "Start" from first TIC and "ReceiverOnHook" from second TIC
For message "onHook" -
[0.0,\infty]$OnHook$ contains [0.0,\infty]$ReceiverPickedUp$

Analysis for state "OnHook" from first TIC and "ReceiverPickedUp" from second TIC
For message "offHook" -
[0.0,\infty]$OffHook$ contains [0.0,\infty]$AcceptDialedNumbers$

Analysis for state "OffHook" from first TIC and "AcceptDialedNumbers" from second TIC
For message "startDialing" -
[0.0,10.0]$Dialing$ contains [0.0,8.0]$DialingDone$

Analysis for state "Dialing" from first TIC and "DialingDone" from second TIC
For message "doneDialing" -
[0.0,\infty]$RouteCall$ contains [0.0,\infty]$AcceptMoney$

Analysis for state "RouteCall" from first TIC and "AcceptMoney" from second TIC
For message "moneyDeposited" -
[0.0,20.0]$MoneyDeposited$ contains [0.0,10.0]$MoneyInSlot$

Analysis for state "MoneyDeposited" from first TIC and "MoneyInSlot" from second TIC
For message "talk" -
[0.0,200.0]$CanTalk$ contains [0.0,180.0]$Talking$

Analysis for state "CanTalk" from first TIC and "Talking" from second TIC
For message "doneTalking" -
[0.0,\infty]$DoneTalking$ contains [0.0,180.0]$Done$

Analysis for state "DoneTalking" from first TIC and "Done" from second TIC

Since no errors were discovered during analysis, these two TICs ARE COMPLETELY COMPATIBLE
=================================
4.2.5. Replaceability Analysis

For replaceability analysis, the temporal interaction contract shown in Fig. 4.2 (called ‘Telephone’) will be compared with the temporal interaction contract shown in Fig 4.6 (called ‘Telephone1’).

The ‘Telephone1’ temporal interaction contract is similar to the temporal interaction contract of ‘Telephone’. The only change is the values of time constraints. These changes are represented in bold font in Fig. 4.6.

The result of partial replaceability analysis for the ‘Telephone’ and ‘Telephone1’ TICs is as shown below.

<table>
<thead>
<tr>
<th>PARTIAL REPLACEABILITY ANALYSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since IntersectionTrace of these two TICs is not null, these two TICs ARE PARTIALLY REPLACEABLE</td>
</tr>
<tr>
<td>The non-null elements of the IntersectionTrace are as follows:</td>
</tr>
<tr>
<td>IntersectionTrace[Start, Start, onHook] = [0.0,Infinity]</td>
</tr>
<tr>
<td>IntersectionTrace[OnHook, OnHook, offHook] = [0.0,Infinity]</td>
</tr>
<tr>
<td>IntersectionTrace[OffHook, OffHook, startDialing] = [0.0,8.0]</td>
</tr>
<tr>
<td>IntersectionTrace[Dialing, Dialing, doneDialing] = [0.0,Infinity]</td>
</tr>
<tr>
<td>IntersectionTrace[RouteCall, RouteCall, noResponse] = [0.0,20.0]</td>
</tr>
<tr>
<td>IntersectionTrace[MoneyDeposited, MoneyDeposited, moneyDeposited] = [0.0,10.0]</td>
</tr>
</tbody>
</table>
Figure 4.6 TIC of ‘Telephone1’ Component (Graphical Representation)
As indicated in the analysis output, since the intersection trace of the ‘Telephone’ and ‘Telephone1’ TIC is not null, these two TICs are partially replaceable. The intersection trace includes all the messages that are common between these two TICs and also the states from which these messages can be activated. For example, consider the following entry in the intersection trace array “IntersectionTrace[OffHook, OffHook, startDialing] = [0.0,8.0]$Dialing$Dialing”. It indicates that the ‘startDialing’ message is common between ‘OffHook’ state from the ‘Telephone’ component and the ‘OffHook’ state from the ‘Telephone1’ component and this message is common for the time period of [0, 8], i.e., 8 seconds after the components enter the ‘OffHook’ state. Similarly, other common messages are listed in the IntersectionTrace array.

The result of complete replaceability analysis for the ‘Telephone’ and ‘Telephone1’ TICs is shown on the next page. The complete replaceability analysis indicates that the ‘Telephone’ and ‘Telephone1’ TICs are not completely replaceable. The time mismatch errors occurred during the analysis are indicated in the output. Consider the error message “ERROR!!!!! [0.0,10.0]$CanTalk does not contain [0.0,20.0]$CanTalk”. This error occurs while analyzing the ‘MoneyDeposited’ state and the ‘talk’ message. The error indicates that the ‘Telephone1’ component can send the ‘talk’ message in the time slice [0, 10] but ‘Telephone’ can send it in the time slice [0, 20]. So there is a time slice [11, 20] between these two components when there is a time mismatch. So there is a time mismatch error reported in the analysis. One possible solution to correct this error would be to reduce the time slice [0, 20] so that [0, 10] can contain it. So if the time slice [0, 20] is changed to [0, 8], then [0, 10] can cover it and that would correct the error.
It should be noted that changing time slice from \([0, 20]\) to \([0, 8]\) implies that, in the ‘Telephone’ component, the value of ‘No Response’ message should be changed from 20 seconds to 8 seconds. Similarly, the error “ERROR!!!!! \([0.0,8.0]\)$Dialing does not contain \([0.0,10.0]\)$Dialing” can be
rectified by changing the time slice from [0, 8] to [0, 12] so that it can contain [0, 10]. This requires changing the value of ‘NoDial’ message in ‘Telephone1’. The old value of 8 seconds should be replaced with 12 seconds. The modified TICs are shown in Fig 4.7 and Fig 4.8.

The partial and complete replaceability analysis for the modified ‘Telephone’ and ‘Telephone1’ TICs is as shown below.

PARTIAL REPLACEABILITY ANALYSIS
================================
Since IntersectionTrace of these two TICs is not null, these two TICs ARE PARTIALLY REPLACEABLE
The non-null elements of the IntersectionTrace are as follows:
IntersectionTrace[Start, Start, onHook] = [0.0,Infinity]$OnHook$OnHook
IntersectionTrace[OnHook, OnHook, offHook] = [0.0,Infinity]$OffHook$OffHook
IntersectionTrace[OffHook, OffHook, startDialing] = [0.0,10.0]$Dialing$Dialing
IntersectionTrace[Dialing, Dialing, doneDialing] = [0.0,Infinity]$RouteCall$RouteCall
IntersectionTrace[RouteCall, RouteCall, noResponse] = [0.0,20.0]$MoneyDeposited$MoneyDeposited
IntersectionTrace[MoneyDeposited, MoneyDeposited, moneyDeposited] = [0.0,8.0]$CanTalk$CanTalk
Figure 4.7 Modified TIC of ‘Telephone’ component (Bold font indicates changes)
Figure 4.8 Modified TIC of ‘Telephone1’ Component (Bold font indicates changes)
As expected, the given changes in both the TICs make them partially as well as completely replaceable.
4.3. **E-commerce System**

This section describes an analysis of an e-commerce system in the context of temporal interaction contracts. The ‘Telephone System’ scenario presented earlier consisted of simply two components executing in a client server manner. However, an e-commerce system is more complex and has a large number of components. These components interact with the user and also may interact with each other to provide certain service required by the user.

Fig 4.9 (and 4.10) shows the class diagram of an e-commerce system under consideration. There is an external entity, user, that interacts with this system. Also, components within the e-commerce system may interact with each other. The e-commerce system consists of the following components: Credit Card, Shipping and Delivery, Customer, Order Form, Shopping Cart, Product and Inventory.

The ‘Product’ component stores the information about the product which is available for sale. The ‘Inventory’ component stores information about the availability of various products (indicated by the ‘describes’ relationship in the class diagram). The ‘Customer’ component stores profile information of various customers (users) of the e-commerce system. The ‘Credit Card’ component stores credit card information of a customer and also provides methods to manipulate this information. Each credit card is associated with a customer (indicated by the ‘owns’ relationship in the class diagram). The ‘Shopping Cart’ component stores information about various products purchased (delivered as well as undelivered) by the customer (indicated by a ‘keeps’ relationship in the class diagram). The ‘Order Form’ component provides the functionality of placing the order for a product. The payment for items in an order form can be done by using the credit card owned by the customer (indicated by ‘payment for’ relationship in the class diagram). The ‘Shipping and Delivery’ component
handles shipping and delivery of the products in order form for which customer has placed an order.
Figure 4.9 Class Diagram of an E-commerce System

Credit Card
- creditCardNumber: Integer
- creditCardType: String
- expirationDate: Date
- verifyCreditCard()
- createCreditCard()
- destroyCreditCard()
- getCreditCard()

Shipping and Delivery
- shippingName: String
- shippingAddress: String
- createShipInfo()
- destroyCreditCard()
- getCreditCard()

Customer
- customerNumber: Integer
- customerName: String
- customerAddress: String
- homePhone: Integer
- workPhone: Integer
- emailAddress: String
- faxNumber: Integer
- updateCustomer()
- register()
- login()
- createCustomer()
- destroyCustomer()
- getCustomer()

Order Form
- orderNumber: Integer
- orderDate: Date
- createOrderForm()
- destroyOrderForm()
- calculateTotal()

Shopping Cart
- shoppingCardId: Integer
- shoppingQuantity: Integer
- displayCart()
- addProductOrder()
- removeProductNumber()
- updateProductOrder()
- getCart()
- checkOut()

A

owns

0..n

1

0..n

1..1

0..1

payment for

requests

keeps

0..n

1

0..n

Field

10..n
The behavior of the components in the e-commerce system can be represented using temporal interaction contracts. Each component in the e-commerce system describes its behavior to the other components with the help of a TIC. Whenever a component wants to interact with other component, it checks whether its own TIC is completely compatible with TIC of the component it is interacting with. If this condition is satisfied, these two components can interact. In the following discussion, TICs for three components, viz. 'Shopping Cart', 'Customer' and 'Customer' are described. TICs for other components can be described in a similar manner.
4.3.1. Temporal Interaction Contract for ‘Shopping Cart’ component

![Diagram of the TIC of Shopping Cart Component]

Figure 4.11 TIC of ‘Shopping Cart’ Component (Graphical Representation)
The temporal interaction contract for the ‘Shopping Cart’ component is shown in Fig. 4.11. When the ‘Shopping Cart’ component begins its execution (i.e. when a user starts placing orders for the products), it enters the ‘Start’ state. Before placing an order for a product, the user searches for that product. When the user begins the search, the ‘Shopping Cart’ component enters the ‘Searching’ state. In this state, the user can add products to the cart, delete products from the cart, or search for other products. After the user is done selecting the products, he/she can get a price quote for the products in the cart. When the quote is requested, the ‘Shopping Cart’ component moves to ‘Calculate Price’ state. If the user wants to search for items again, then the ‘Shopping Cart’ component switches back to the ‘Searching’ state. After getting a price quote, the user can either explicitly cancel it or the quote become invalid after an expiry period of 60 minutes. In both the cases, ‘Shopping Cart’ component changes its state to ‘Cancel’ and terminates its execution.

If the user is satisfied with the quoted price of the products, he/she can place an order. When user places an order, the state of the ‘Shopping Cart’ component changes to ‘Can be cancelled’. In this state, the user can still cancel the order. However, if user does not cancel the order within 80 minutes, then the ‘Shopping Cart’ component automatically moves to the ‘Place Order’ state. In this state, an order is placed which can not be cancelled. This terminates the execution of the component.

4.3.2. Temporal Interaction Contract for ‘Credit Card’ component

Fig 4.12 shows the temporal interaction contract of a ‘Credit Card’ component. This component handles the credit card information of a particular user. It provides methods to create new credit cards and do transactions with the help of credit cards.
Figure 4.12 TIC of ‘Credit Card’ Component (Graphical Representation)

The ‘Credit Card’ component begins its execution in the ‘Start’ state. When a new credit card is created, it moves to the ‘Credit Card Created’ state. If the user wants to destroy the credit card, then the ‘Credit Card’ component enters the ‘Credit Card Destroyed’ state. However, the ‘Credit Card’ component automatically moves to the ‘Credit Card Expires’ state when the credit card expires. If the user wants to perform certain transactions using the credit card,
the ‘Credit Card’ component enters the ‘Transaction’ state. In this state, user can perform transactions using this credit card. But again after the credit card expires, the ‘Credit Card’ component automatically moves to ‘Credit Card Expires’ state. The execution of the ‘Credit Card’ component terminates when it reaches either ‘Credit Card Expires’ state or ‘Credit Card Destroyed’ state.

4.3.3. Temporal Interaction Contract for ‘Customer’ component

The temporal interaction contract of the ‘Customer’ component is shown in the Fig. 4.13 below. The ‘Customer’ component stores information about customers of the e-commerce system. It provides the functionality required to create, maintain and manipulate customer accounts.

The ‘Customer’ component starts its execution in the ‘Start’ state. When a user wants to register, the ‘Customer’ component enters the ‘Customer Registered’ state. When the user creates a profile, the ‘Customer’ component enters the ‘Account Created’ state. However, if the customer does not access his/her account for certain time period, the ‘Customer’ account automatically enters the ‘Delete Account’ state. In this state, the corresponding customer account is deleted and the ‘Customer’ component terminates its execution.

Once the account is created, the user can update his/her account, view account information and delete their account. If the user requests to view the account information, the ‘Customer’ component enters the ‘Display Info’ state. If the user requests to delete the account, the ‘Customer’ component enters the ‘Delete Account’ state and terminates its execution.
Figure 4.13 TIC of 'Customer' Component (Graphical Representation)
4.3.4. Compatibility and Replaceability Analysis for E-commerce System

After the TICs for all the components in the e-commerce system are specified, the compatibility and replaceability analyses can be carried out in a way similar to the analysis for ‘Telephone’ system. Consider the ‘Credit Card’ component for compatibility and replaceability analysis.

The ‘Credit Card’ component interacts with two other components in the system: the ‘Customer’ component which creates a credit card for the customer, and ‘Shopping Cart’ component, which places an order for the products. The ‘Credit Card’ component takes this order as a transaction and does further processing to charge the account of the customer who owns this credit card.

Figure 4.14 ‘Credit Card’ Component With Its ‘Environment’
As shown in Fig. 4.14., the ‘Shopping Cart’ and ‘Customer’ components together form the ‘Environment’ with which the ‘Credit Card’ component interacts. In reality, the ‘Credit Card’ component reacts with both the ‘Customer’ component and the ‘Shopping Card’ component (as shown by dotted arrows), but their resultant interaction can be thought of as that between the ‘Credit Card’ component and ‘Environment’ (as shown by the solid arrow). So, while testing compatibility between ‘Credit Card’ and the components with which it interacts, the compatibility between the ‘Credit Card’ TIC and the ‘Environment’ TIC should be tested.

The TIC of the ‘Environment’ is as shown in the Fig 4.15. The complete and partial compatibility analysis for ‘Credit Card’ and ‘Environment’ is as shown below.

<table>
<thead>
<tr>
<th>PARTIAL COMPATIBILITY ANALYSIS</th>
</tr>
</thead>
</table>

Since InteractionTrace of these two TICs is not null, these two TICs ARE PARTIALLY COMPATIBLE

The non-null elements of the InteractionTrace are as follows:

- InteractionTrace[Start, CreateCard, createCreditCard] = [0.0, Infinity] $CreditCardCreated$TransactionRequest
- InteractionTrace[CreditCardCreated, TransactionRequest, destroy] = [0.0, 40.0] $CreditCardDestroyed$TransactionActive
- InteractionTrace[CreditCardCreated, TransactionRequest, destroy] = [0.0, 40.0] $Transaction$TransactionActive
Create Card

createCreditCard()

[OUT]

Transaction Request

startTransaction()

[OUT]

Shopping Period Expires: 40 mins

Shopping Done

Transaction Active

Shopping Period Expires: 40 mins

Figure 4.15 TIC For the ‘Environment’
Thus the ‘Credit Card’ component is completely compatible with its environment. Similar analyses can be carried out for all other interacting components in the e-commerce system.

### 4.4. TIC Simulation

As discussed earlier in Section 4.1, TICP also includes a facility to simulate the component behavior based on its temporal interaction contract. The simulator component of TICP is termed as TICS (Temporal Interaction Contract Simulator). The TICS contains functionality to simulate server components (like the ‘Telephone’ component) and client components (like the ‘Dialer’ component). By running server and client simulations in parallel, one can check if there are time-related mismatches in their interactions. TICS output can be viewed in conjunction with partial/complete compatibility analysis output. The analysis
output presents the results of static analysis done on TIC information while TICS output actually simulates the runtime behavior of a component (based on its TIC) thus giving a confirmation of the results obtained in the matching analysis.

4.4.1. ‘Telephone’ and ‘Dialer’ component

The ‘Telephone’ and ‘Dialer’ TICs (shown in Fig. 4.2 and 4.3) are fed to the TICS for the interaction simulation. The output of simulation of the interaction between ‘Telephone’ and ‘Dialer’ component is as shown below.

```
Starting the simulation for Telephone component
==============================================
current state is Start
current message received is onHook
current state is OnHook
current message received is offHook
current state is OffHook
Duration is10
current state is PutOnHook
End state is reached
current message is startDialing
current state is ERROR!!! The state "PutOnHook" can not receive the message "startDialing"
java.lang.ArrayIndexOutOfBoundsException: -1
    at code.Server_simulator.hasTemporal(Server_simulator.java:59)
    at code.Server_simulator.main(Server_simulator.java:280)
java.lang.ArrayIndexOutOfBoundsException: -1
    at code.Server_simulator.hasTemporal(Server_simulator.java:59)
at code.Server_simulator$1$RemindTask.run(Server_simulator.java:162)
at java.util.TimerThread.mainLoop(Timer.java:432)
at java.util.TimerThread.run(Timer.java:382)
Exception in thread "main"
```
As indicated in the ‘Telephone’ component simulation, the error occurs during its execution because when the ‘Telephone’ component enters the ‘Put On Hook’ state, it receives the ‘startDialing’ message but the ‘Telephone’ component cannot process this message in the ‘Put On Hook’ state. This indicates that the ‘Telephone’ and ‘Dialer’ component are not completely compatible. However, they are partially compatible as some interactions occur between them. For example: the ‘Dialer’ component sends an ‘onHook’ message which is received by the ‘Telephone’ component. Other possible interactions are also indicated in the simulation output. Thus, the simulation indicates that the ‘Telephone’ and ‘Dialer’ component are partially compatible but they are not completely compatible. This matches exactly with the results obtained from the matching analysis done earlier.

4.4.2. Modified ‘Telephone’ and ‘Dialer’ component

The modified ‘Telephone’ and ‘Dialer’ TICs (shown in Fig. 4.4 and 4.5) are fed to the TICS for the interaction simulation. The output of simulation of the interaction between modified ‘Telephone’ and ‘Dialer’ component is as shown below.

Starting the simulation for Dialer component
==============================================
current state is ReceiverOnHook
current message sent is onHook
current state is ReceiverPickedUp
current message sent is offHook
current state is AcceptDialedNumbers
current message sent is startDialing
current state is DialingDone
current message sent is doneDialing
current state is AcceptMoney
current message sent is moneyDeposited
current state is MoneyInSlot
current message sent is talk
Starting the simulation for Modified Telephone component

- Current state is Start
- Current message received is onHook
- Current state is OnHook
- Current message received is offHook
- Current state is OffHook
- Duration is 10
- Current message is startDialing
- Current state is Dialing
- Current message received is doneDialing
- Current state is RouteCall
- Duration is 20
- Current message is moneyDeposited
- Current state is MoneyDeposited
- Duration is 200
- Current message is talk
- Current state is CanTalk
- Current message received is doneTalking
- Current state is DoneTalking
- End state is reached

Starting the simulation for Modified Dialer component

- Current state is ReceiverOnHook
- Current message sent is onHook
- Current state is ReceiverPickedUp
- Current message sent is offHook
- Current state is AcceptDialedNumbers
- Current message sent is startDialing
- Current state is DialingDone
- Current message sent is doneDialing
- Current state is AcceptMoney
- Current message sent is moneyDeposited
- Current state is MoneyInSlot
- Current message sent is talk
- Current state is Talking
- Current message sent is doneTalking
- Current state is Done
- End state is reached
As indicated in the 'Modified Telephone' component simulation, there are no errors indicated and the 'Modified Telephone' component reaches its end state and terminates its execution. Similarly, the 'Modified Dialer' component reaches end state and terminates its execution without any errors. This indicates that the 'Modified Telephone' and 'Modified Dialer' component are completely compatible. Also, they are partially compatible as some interactions occur between them. For example: the 'Dialer' component sends the 'onHook' message which is received by the 'Telephone' component. Other possible interactions are also indicated in the simulation output. Thus, the simulation indicates that the 'Modified Telephone' and 'Modified Dialer' component are partially compatible as well as completely compatible. This matches exactly with the results obtained from the matching analysis done earlier.

4.5. Summary

This chapter provides an example for illustrating the key concepts of the thesis. It provides a description of the system under consideration and also gives the temporal interaction contract of the components in the system. The chapter also illustrates the matching of the temporal interaction contracts and how the errors occurred during match can be corrected. The simulator outputs are shown to be consistent with the matching analysis. The next chapter summarizes the thesis by providing the conclusions of the research work, the contributions of the thesis and a few of the possible future extensions.
CHAPTER 5. CONCLUSION AND FUTURE WORK

This thesis presented a mechanism for specifying the components temporal ordering constraints in the form of temporal interaction contracts and mechanisms to match the contracts for compatibility and replaceability. The next section provides the conclusions of the research work. Section 5.2 presents an overview of the mechanism for specifying the contracts and defining the matching criteria for compatibility and replaceability. The section 5.3 gives possible future extensions to the research work, and the last section summarizes this thesis.

5.1. Conclusions of the Research

The following are the conclusions drawn from the research work:

1. The temporal interaction contract matching analysis reveals time mismatch errors. These errors can be rectified to make the resultant components completely compatible/replaceable with each other.

2. The temporal interaction contract matching analysis detects time errors before the two interacting components are actually connected. This early analysis saves the time of the system developer.

3. Since complete temporal interaction contract matching analysis ensures that two interacting components are without time mismatch errors, it is essential for the reliable (from the temporal perspective) operation of DCS.
5.2. Overview of Temporal Interaction Contracts

For selecting the components to form a system from the component library and to predict the behavior of the resultant system, the component developers need to specify the component's interface at each of the four levels namely, the syntactic, the semantic, the synchronization and the QoS levels and create contracts at each of the levels. [ZAR96] provides a mechanism for specifying the component's interface at the syntactic and the semantic levels. This thesis provides the mechanism to specify the temporal ordering constraints at the level of the component's methods [Chapter 3]. These specifications are called ‘Temporal Interaction Contracts’. The thesis also provides a mechanism to match the temporal interaction contracts of two components. Matching of the contracts is needed as the component library consists of a large number of components and a search is required to find a component that provides the required temporal functionality. Thus, the temporal interaction contracts help users match the specifications and make a selection decision amongst the components with the same temporal functionality. It also helps the users predict the temporal behavior of the resultant system before actually integrating the components. The temporal behavior of the resultant system then can be verified against the temporal behavior of the desired system.

The following are the features of the temporal interaction contracts:

1. The temporal interaction contracts reflect the temporal ordering constraints associated with the execution of the component. These constraints are expressed at a high level of abstraction (i.e., at the level of component methods).

2. The partial match gives a relaxed criterion for matching two temporal interaction contracts.

3. The complete match gives a stringent criterion for matching two temporal interaction contracts.
4. The contracts and the matches have been provided irrespective of the component model and the implementation language of the component and hence, are applicable to a variety of software components.

5.3. Contributions of this Thesis

The contributions of this thesis are as follows:

1. Provision of a mechanism to create component contracts for temporal constraints on component interactions.
2. Definition of a set of matching criteria for matching temporal interaction contracts of the components.
3. Provision of a mechanism to match the temporal interaction contracts of the components.

5.4. Future Work

Several future extensions to this research work are possible and few of them have been suggested below:

1. The formal representation of the temporal interaction contracts is done manually. For complex system it might be time consuming and error prone. Moreover, the user has to learn and understand the meaning of various XML tags to represent the contracts. Hence, a system to automatically generate these formal representations would make the job task easier.
2. The contracts and the matching criteria can be incorporated within the UniFrame Resource Discovery Service to refine the searching of components.
3. A runtime monitoring system can be designed to verify the results of matching algorithms against the behavior of components when they are actually interacting with each other.
4. At present, timing constraints having a fixed value are considered for analysis. However, TLA operators can be included in the temporal interaction contracts to expand their expressive power.

5.5. Summary

The thesis has presented temporal interaction contracts for software components to facilitate the selection of components from a component library. The matching criteria defined for the compatibility and replaceability aids in the creation of high-confidence DCS. The thesis also provides a case study of the ‘Telephone’ and ‘E-commerce’ systems to validate the research work.
LIST OF REFERENCES


Appendix A. XML Specification of TICs

The Appendix consists of the XML representations of the TICs used for the Case Study.

XML Schema for TIC:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xs:element name="Component">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="ComponentName"/>
                <xs:element ref="TemporalInteractionContract"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ComponentName" type="xs:NCName"/>
    <xs:element name="TemporalInteractionContract">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="States"/>
                <xs:element ref="Messages"/>
                <xs:element ref="Transitions"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="States">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="StartState"/>
                <xs:element maxOccurs="unbounded" ref="State"/>
                <xs:element maxOccurs="unbounded" ref="EndState"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="StartState" type="xs:NCName"/>
    <xs:element name="State" type="xs:NCName"/>
    <xs:element name="EndState" type="xs:NCName"/>
    <xs:element name="Messages">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="Message"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Messages" type="xs:NCName"/>
</xs:schema>
```
(Continued) XML Schema for TIC:

```xml
<xs:element name="Message" type="xs:NCName"/>
<xs:element name="Transitions">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="Transition"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Transition">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="SourceState"/>
      <xs:element ref="DestinationState"/>
      <xs:element ref="TransitionMessage"/>
      <xs:element ref="Direction"/>
      <xs:element ref="TemporalConstraint"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="SourceState" type="xs:NCName"/>
<xs:element name="DestinationState" type="xs:NCName"/>
<xs:element name="TransitionMessage" type="xs:NCName"/>
<xs:element name="Direction" type="xs:NCName"/>
<xs:element name="TemporalConstraint" type="xs:integer"/>
</xs:schema>
```
TIC of the ‘Telephone’ component:

```xml
<Component>
  <ComponentName>TELEPHONE</ComponentName>
  <TemporalInteractionContract>
    <States>
      <StartState>Start</StartState>
      <State>OnHook</State>
      <State>OffHook</State>
      <EndState>PutOnHook</EndState>
      <State>Dialing</State>
      <State>RouteCall</State>
      <State>MoneyDeposited</State>
      <EndState>NoMoney</EndState>
      <State>CanTalk</State>
      <EndState>DoneTalking</EndState>
      <EndState>ReturnMoney</EndState>
    </States>
    <Messages>
      <Message>onHook</Message>
      <Message>offHook</Message>
      <Message>startDialing</Message>
      <Message>noDial</Message>
      <Message>doneDialing</Message>
      <Message>noResponse</Message>
      <Message>moneyDeposited</Message>
      <Message>talk</Message>
      <Message>doneTalking</Message>
      <Message>noMoney</Message>
    </Messages>
  </TemporalInteractionContract>
</Component>
```
(Continued) TIC of the ‘Telephone’ component:

```
<Transitions>
  <Transition>
    <SourceState>Start</SourceState>
    <DestinationState>OnHook</DestinationState>
    <TransitionMessage>onHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OnHook</SourceState>
    <DestinationState>OffHook</DestinationState>
    <TransitionMessage>offHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>PutOnHook</DestinationState>
    <TransitionMessage>noDial</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>10</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>Dialing</DestinationState>
    <TransitionMessage>startDialing</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>Dialing</SourceState>
    <DestinationState>RouteCall</DestinationState>
    <TransitionMessage>doneDialing</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>RouteCall</SourceState>
    <DestinationState>NoMoney</DestinationState>
    <TransitionMessage>noMoney</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>20</TemporalConstraint>
  </Transition>
</Transitions>
```
(Continued) TIC of the ‘Telephone’ component:

<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>CanTalk</DestinationState>
  <TransitionMessage>talk</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
<Transition>
  <SourceState>CanTalk</SourceState>
  <DestinationState>DoneTalking</DestinationState>
  <TransitionMessage>doneTalking</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>ReturnMoney</DestinationState>
  <TransitionMessage>noResponse</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>20</TemporalConstraint>
</Transition>
</Transitions>
</Component>
**TIC of the ‘Dialer’ component:**

```xml
<Component>
  <ComponentName>DIALER</ComponentName>
  <TemporalInteractionContract>

    <States>
      <StartState>ReceiverOnHook</StartState>
      <State>ReceiverPickedUp</State>
      <State>AcceptDialedNumbers</State>
      <EndState>End</EndState>
      <State>DialingDone</State>
      <State>AcceptMoney</State>
      <EndState>SlotEmpty</EndState>
      <State>MoneyInSlot</State>
      <State>Talking</State>
      <EndState>Done</EndState>
      <EndState>TimeUp</EndState>
    </States>

    <Messages>
      <Message>onHook</Message>
      <Message>offHook</Message>
      <Message>startDialing</Message>
      <Message>noKeyPressed</Message>
      <Message>startDialing</Message>
      <Message>doneDialing</Message>
      <Message>noMoneyInSlot</Message>
      <Message>moneyDeposited</Message>
      <Message>talk</Message>
      <Message>doneTalking</Message>
      <Message>timeUp</Message>
    </Messages>

    <Transitions>
      <Transition>
        <SourceState>ReceiverOnHook</SourceState>
        <DestinationState>ReceiverPickedUp</DestinationState>
        <TransitionMessage>onHook</TransitionMessage>
        <Direction>OUT</Direction>
        <TemporalConstraint>NO</TemporalConstraint>
      </Transition>

      <Transition>
        <SourceState>ReceiverPickedUp</SourceState>
        <DestinationState>AcceptDialedNumbers</DestinationState>
        <TransitionMessage>offHook</TransitionMessage>
        <Direction>OUT</Direction>
        <TemporalConstraint>NO</TemporalConstraint>
      </Transition>
    </Transitions>

  </TemporalInteractionContract>
</Component>
```
(Continued) TIC of the ‘Dialer’ component:

<table>
<thead>
<tr>
<th>Transition</th>
<th>SourceState</th>
<th>DestinationState</th>
<th>TransitionMessage</th>
<th>Direction</th>
<th>TemporalConstraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Transition&gt;</td>
<td>AcceptDialledNumbers</td>
<td>End</td>
<td>noKeyPressed</td>
<td>OUT</td>
<td>15</td>
</tr>
<tr>
<td>&lt;Transition&gt;</td>
<td>AcceptDialledNumbers</td>
<td>DialingDone</td>
<td>startDialing</td>
<td>OUT</td>
<td>NO</td>
</tr>
<tr>
<td>&lt;Transition&gt;</td>
<td>DialingDone</td>
<td>AcceptMoney</td>
<td>doneDialing</td>
<td>OUT</td>
<td>NO</td>
</tr>
<tr>
<td>&lt;Transition&gt;</td>
<td>AcceptMoney</td>
<td>SlotEmpty</td>
<td>noMoneyInSlot</td>
<td>OUT</td>
<td>10</td>
</tr>
<tr>
<td>&lt;Transition&gt;</td>
<td>AcceptMoney</td>
<td>MoneyInSlot</td>
<td>moneyDeposited</td>
<td>OUT</td>
<td>NO</td>
</tr>
<tr>
<td>&lt;Transition&gt;</td>
<td>MoneyInSlot</td>
<td>Talking</td>
<td>talk</td>
<td>OUT</td>
<td>NO</td>
</tr>
<tr>
<td>&lt;Transition&gt;</td>
<td>MoneyInSlot</td>
<td>TimeUp</td>
<td>timeUp</td>
<td>OUT</td>
<td>180</td>
</tr>
</tbody>
</table>
(Continued) TIC of the ‘Dialer’ component:

```
<Transition>
  <SourceState>Talking</SourceState>
  <DestinationState>TimeUp</DestinationState>
  <TransitionMessage>timeUp</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>180</TemporalConstraint>
</Transition>
```

```
<Transition>
  <SourceState>Talking</SourceState>
  <DestinationState>Done</DestinationState>
  <TransitionMessage>doneTalking</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
```

```
</Transitions>
</TemporalInteractionContract>
```

Modified TIC of the ‘Telephone’ component:

```
<Component>
  <ComponentName>TELEPHONE</ComponentName>
  <TemporalInteractionContract>
    <States>
      <StartState>Start</StartState>
      <State>OnHook</State>
      <State>OffHook</State>
      <EndState>PutOnHook</EndState>
      <State>Dialing</State>
      <State>RouteCall</State>
      <State>MoneyDeposited</State>
      <EndState>NoMoney</EndState>
      <State>CanTalk</State>
      <EndState>DoneTalking</EndState>
      <EndState>ReturnMoney</EndState>
    </States>
  </TemporalInteractionContract>
</Component>
```
(Continued) Modified TIC of the ‘Telephone’ component:

```xml
<Messages>
  <Message>onHook</Message>
  <Message>offHook</Message>
  <Message>startDialing</Message>
  <Message>noDial</Message>
  <Message>doneDialing</Message>
  <Message>noResponse</Message>
  <Message>moneyDeposited</Message>
  <Message>talk</Message>
  <Message>doneTalking</Message>
  <Message>noMoney</Message>
</Messages>

<Transitions>
  <Transition>
    <SourceState>Start</SourceState>
    <DestinationState>OnHook</DestinationState>
    <TransitionMessage>onHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OnHook</SourceState>
    <DestinationState>OffHook</DestinationState>
    <TransitionMessage>offHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>PutOnHook</DestinationState>
    <TransitionMessage>noDial</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>10</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>Dialing</DestinationState>
    <TransitionMessage>startDialing</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
</Transitions>
```
(Continued) Modified TIC of the ‘Telephone’ component:

```xml
<Transition>
  <SourceState>Dialing</SourceState>
  <DestinationState>RouteCall</DestinationState>
  <TransitionMessage>doneDialing</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
<Transition>
  <SourceState>RouteCall</SourceState>
  <DestinationState>NoMoney</DestinationState>
  <TransitionMessage>noMoney</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>20</TemporalConstraint>
</Transition>
<Transition>
  <SourceState>RouteCall</SourceState>
  <DestinationState>MoneyDeposited</DestinationState>
  <TransitionMessage>moneyDeposited</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>CanTalk</DestinationState>
  <TransitionMessage>talk</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
<Transition>
  <SourceState>CanTalk</SourceState>
  <DestinationState>DoneTalking</DestinationState>
  <TransitionMessage>doneTalking</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>ReturnMoney</DestinationState>
  <TransitionMessage>noResponse</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>200</TemporalConstraint>
</Transition>
</Transitions>
</TemporalInteractionContract>
</Component>
```
(Continued) Modified TIC of the ‘Dialer’ component:

```xml
<Component>
    <ComponentName>DIALER</ComponentName>
    <TemporalInteractionContract>

    <States>
        <StartState>ReceiverOnHook</StartState>
        <State>ReceiverPickedUp</State>
        <State>AcceptDialedNumbers</State>
        <EndState>End</EndState>
        <State>DialingDone</State>
        <State>AcceptMoney</State>
        <EndState>SlotEmpty</EndState>
        <State>MoneyInSlot</State>
        <State>Talking</State>
        <EndState>Done</EndState>
        <EndState>TimeUp</EndState>
    </States>

    <Messages>
        <Message>onHook</Message>
        <Message>offHook</Message>
        <Message>startDialing</Message>
        <Message>noKeyPressed</Message>
        <Message>startDialing</Message>
        <Message>doneDialing</Message>
        <Message>noMoneyInSlot</Message>
        <Message>moneyDeposited</Message>
        <Message>talk</Message>
        <Message>doneTalking</Message>
        <Message>timeUp</Message>
    </Messages>

    <Transitions>
        <Transition>
            <SourceState>ReceiverOnHook</SourceState>
            <DestinationState>ReceiverPickedUp</DestinationState>
            <TransitionMessage>onHook</TransitionMessage>
            <Direction>OUT</Direction>
            <TemporalConstraint>NO</TemporalConstraint>
        </Transition>
        <Transition>
            <SourceState>ReceiverPickedUp</SourceState>
            <DestinationState>AcceptDialedNumbers</DestinationState>
            <TransitionMessage>offHook</TransitionMessage>
            <Direction>OUT</Direction>
            <TemporalConstraint>NO</TemporalConstraint>
        </Transition>
    </Transitions>

</Component>
```
(Continued) Modified TIC of the ‘Dialer’ component:

```
<Transition>
  <SourceState>AcceptDialedNumbers</SourceState>
  <DestinationState>End</DestinationState>
  <TransitionMessage>noKeyPressed</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>8</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>AcceptDialedNumbers</SourceState>
  <DestinationState>DialingDone</DestinationState>
  <TransitionMessage>startDialing</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>DialingDone</SourceState>
  <DestinationState>AcceptMoney</DestinationState>
  <TransitionMessage>doneDialing</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>AcceptMoney</SourceState>
  <DestinationState>SlotEmpty</DestinationState>
  <TransitionMessage>noMoneyInSlot</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>10</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>AcceptMoney</SourceState>
  <DestinationState>MoneyInSlot</DestinationState>
  <TransitionMessage>moneyDeposited</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>MoneyInSlot</SourceState>
  <DestinationState>Talking</DestinationState>
  <TransitionMessage>talk</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>MoneyInSlot</SourceState>
  <DestinationState>TimeUp</DestinationState>
  <TransitionMessage>timeUp</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>180</TemporalConstraint>
</Transition>
```
(Continued) Modified TIC of the ‘Dialer’ component:

```xml
<Transition>
  <SourceState>Talking</SourceState>
  <DestinationState>TimeUp</DestinationState>
  <TransitionMessage>timeUp</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>180</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>Talking</SourceState>
  <DestinationState>Done</DestinationState>
  <TransitionMessage>doneTalking</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
</Transitions>
</TemporalInteractionContract>

<Component>
  <ComponentName>TELEPHONE</ComponentName>
  <TemporalInteractionContract>
    <States>
      <StartState>Start</StartState>
      <State>OnHook</State>
      <State>OffHook</State>
      <EndState>PutOnHook</EndState>
      <State>Dialing</State>
      <State>RouteCall</State>
      <State>MoneyDeposited</State>
      <EndState>NoMoney</EndState>
      <State>CanTalk</State>
      <EndState>DoneTalking</EndState>
      <EndState>ReturnMoney</EndState>
    </States>
  </TemporalInteractionContract>
</Component>
```
(Continued) TIC of the ‘Telephone1’ component:

```xml
<Message>onHook</Message>
<Message>offHook</Message>
<Message>startDialing</Message>
<Message>noDial</Message>
<Message>doneDialing</Message>
<Message>noResponse</Message>
<Message>moneyDeposited</Message>
<Message>talk</Message>
<Message>doneTalking</Message>
<Message>noMoney</Message>

<Transitions>
  <Transition>
    <SourceState>Start</SourceState>
    <DestinationState>OnHook</DestinationState>
    <TransitionMessage>onHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OnHook</SourceState>
    <DestinationState>OffHook</DestinationState>
    <TransitionMessage>offHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>PutOnHook</DestinationState>
    <TransitionMessage>noDial</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>8</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>Dialing</DestinationState>
    <TransitionMessage>startDialing</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
</Transitions>
```
(Continued) TIC of the ‘Telephone1’ component:

```xml
<Transition>
    <SourceState>Dialing</SourceState>
    <DestinationState>RouteCall</DestinationState>
    <TransitionMessage>doneDialing</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
    <SourceState>RouteCall</SourceState>
    <DestinationState>NoMoney</DestinationState>
    <TransitionMessage>noMoney</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>30</TemporalConstraint>
</Transition>

<Transition>
    <SourceState>RouteCall</SourceState>
    <DestinationState>MoneyDeposited</DestinationState>
    <TransitionMessage>moneyDeposited</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
    <SourceState>MoneyDeposited</SourceState>
    <DestinationState>CanTalk</DestinationState>
    <TransitionMessage>talk</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
    <SourceState>MoneyDeposited</SourceState>
    <DestinationState>ReturnMoney</DestinationState>
    <TransitionMessage>noResponse</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>10</TemporalConstraint>
</Transition>

</Transitions>
</TemporalInteractionContract>
</Component>
```
TIC of the modified ‘Telephone’ component (for Replaceability):

```xml
<Component>
  <ComponentName>TELEPHONE</ComponentName>
  <TemporalInteractionContract>
    <States>
      <StartState>Start</StartState>
      <State>OnHook</State>
      <State>OffHook</State>
      <EndState>PutOnHook</EndState>
      <State>Dialing</State>
      <State>RouteCall</State>
      <State>MoneyDeposited</State>
      <EndState>NoMoney</EndState>
      <State>CanTalk</State>
      <EndState>DoneTalking</EndState>
      <EndState>ReturnMoney</EndState>
    </States>
    <Messages>
      <Message>onHook</Message>
      <Message>offHook</Message>
      <Message>startDialing</Message>
      <Message>noDial</Message>
      <Message>doneDialing</Message>
      <Message>noResponse</Message>
      <Message>moneyDeposited</Message>
      <Message>talk</Message>
      <Message>doneTalking</Message>
      <Message>noMoney</Message>
    </Messages>
    <Transitions>
      <Transition>
        <SourceState>Start</SourceState>
        <DestinationState>OnHook</DestinationState>
        <TransitionMessage>onHook</TransitionMessage>
        <Direction>IN</Direction>
        <TemporalConstraint>NO</TemporalConstraint>
      </Transition>
      <Transition>
        <SourceState>OnHook</SourceState>
        <DestinationState>OffHook</DestinationState>
        <TransitionMessage>offHook</TransitionMessage>
        <Direction>IN</Direction>
        <TemporalConstraint>NO</TemporalConstraint>
      </Transition>
    </Transitions>
  </TemporalInteractionContract>
</Component>
```
(Continued) TIC of the modified ‘Telephone’ component (for Replaceability):

```xml
<Transition>
  <SourceState>OffHook</SourceState>
  <DestinationState>PutOnHook</DestinationState>
  <TransitionMessage>noDial</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>10</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>OffHook</SourceState>
  <DestinationState>Dialing</DestinationState>
  <TransitionMessage>startDialing</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>Dialing</SourceState>
  <DestinationState>RouteCall</DestinationState>
  <TransitionMessage>doneDialing</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>RouteCall</SourceState>
  <DestinationState>NoMoney</DestinationState>
  <TransitionMessage>noMoney</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>20</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>RouteCall</SourceState>
  <DestinationState>MoneyDeposited</DestinationState>
  <TransitionMessage>moneyDeposited</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>CanTalk</DestinationState>
  <TransitionMessage>talk</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>
```
(Continued) TIC of the modified ‘Telephone’ component (for Replaceability):

```xml
<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>ReturnMoney</DestinationState>
  <TransitionMessage>noResponse</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>8</TemporalConstraint>
</Transition>
</Transitions>
</TemporalInteractionContract>
</Component>
```

TIC of the modified ‘Telephone1’ component (for Replaceability):

```xml
<Component>
  <ComponentName>TELEPHONE</ComponentName>
  <TemporalInteractionContract>
    <States>
      <StartState>Start</StartState>
      <State>OnHook</State>
      <State>OffHook</State>
      <EndState>PutOnHook</EndState>
      <State>Dialing</State>
      <State>RouteCall</State>
      <State>MoneyDeposited</State>
      <EndState>NoMoney</EndState>
      <State>CanTalk</State>
      <EndState>DoneTalking</EndState>
      <EndState>ReturnMoney</EndState>
    </States>
  </TemporalInteractionContract>
</Component>
```
(Continued) TIC of the modified ‘Telephone1’ component (for Replaceability):

```xml
<Message>onHook</Message>
<Message>offHook</Message>
<Message>startDialing</Message>
<Message>noDial</Message>
<Message>doneDialing</Message>
<Message>noResponse</Message>
<Message>moneyDeposited</Message>
<Message>talk</Message>
<Message>doneTalking</Message>
<Message>noMoney</Message>
</Messages>

<Transitions>
  <Transition>
    <SourceState>Start</SourceState>
    <DestinationState>OnHook</DestinationState>
    <TransitionMessage>onHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OnHook</SourceState>
    <DestinationState>OffHook</DestinationState>
    <TransitionMessage>offHook</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>PutOnHook</DestinationState>
    <TransitionMessage>noDial</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>12</TemporalConstraint>
  </Transition>
  <Transition>
    <SourceState>OffHook</SourceState>
    <DestinationState>Dialing</DestinationState>
    <TransitionMessage>startDialing</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>
</Transitions>
```
(Continued) TIC of the modified ‘Telephone1’ component (for Replaceability):

```xml
<Transition>
  <SourceState>Dialing</SourceState>
  <DestinationState>RouteCall</DestinationState>
  <TransitionMessage>doneDialing</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>RouteCall</SourceState>
  <DestinationState>NoMoney</DestinationState>
  <TransitionMessage>noMoney</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>30</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>RouteCall</SourceState>
  <DestinationState>MoneyDeposited</DestinationState>
  <TransitionMessage>moneyDeposited</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>CanTalk</DestinationState>
  <TransitionMessage>talk</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>MoneyDeposited</SourceState>
  <DestinationState>ReturnMoney</DestinationState>
  <TransitionMessage>noResponse</TransitionMessage>
  <Direction>IN</Direction>
  <TemporalConstraint>10</TemporalConstraint>
</Transition>
</TemporalInteractionContract>
</Component>
```
TIC of the ‘Credit Card’ component:

```xml
<Component>
  <ComponentName>CREDITCARD</ComponentName>
  <TemporalInteractionContract>

    <States>
      <StartState>Start</StartState>
      <State>CreditCardCreated</State>
      <State>Transaction</State>
      <EndState>CreditCardExpires</EndState>
      <EndState>CreditCardDestroyed</EndState>
    </States>

    <Messages>
      <Message>createCreditCard</Message>
      <Message>destroy</Message>
      <Message>startTransaction</Message>
      <Message>cardExpires</Message>
    </Messages>

  </TemporalInteractionContract>
</Component>
```
(Continued) TIC of the ‘Credit Card’ component:

```
<Transitions>

  <Transition>
    <SourceState>Start</SourceState>
    <DestinationState>CreditCardCreated</DestinationState>
    <TransitionMessage>createCreditCard</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>

  <Transition>
    <SourceState>CreditCardCreated</SourceState>
    <DestinationState>Transaction</DestinationState>
    <TransitionMessage>startTransaction</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>

  <Transition>
    <SourceState>CreditCardCreated</SourceState>
    <DestinationState>CreditCardDestroyed</DestinationState>
    <TransitionMessage>destroy</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
  </Transition>

</Transitions>
```
(Continued) TIC of the ‘Credit Card’ component:

<Transition>
    <SourceState>CreditCardCreated</SourceState>
    <DestinationState>CreditCardExpires</DestinationState>
    <TransitionMessage>cardExpires</TransitionMessage>
    <Direction>NO</Direction>
    <TemporalConstraint>4000</TemporalConstraint>
</Transition>

<Transition>
    <SourceState>Transaction</SourceState>
    <DestinationState>CreditCardDestroyed</DestinationState>
    <TransitionMessage>destroy</TransitionMessage>
    <Direction>IN</Direction>
    <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
    <SourceState>Transaction</SourceState>
    <DestinationState>CreditCardExpires</DestinationState>
    <TransitionMessage>cardExpires</TransitionMessage>
    <Direction>NO</Direction>
    <TemporalConstraint>4000</TemporalConstraint>
</Transition>

</Transitions>
</TemporalInteractionContract>
</Component>
TIC of the ‘Environment’:

```xml
لزمونبم <Component>
  <ComponentName>ENVIRONMENT</ComponentName>
  <TemporalInteractionContract>
    <States>
      <StartState>CreateCard</StartState>
      <State>TransactionRequest</State>
      <State>TransactionActive</State>
      <EndState>ShoppingDone</EndState>
    </States>
    <Messages>
      <Message>createCreditCard</Message>
      <Message>destroy</Message>
      <Message>startTransaction</Message>
      <Message>cardExpires</Message>
      <Message>shoppingPeriodExpires</Message>
    </Messages>
    <Transitions>
      <Transition>
        <SourceState>CreateCard</SourceState>
        <DestinationState>TransactionRequest</DestinationState>
        <TransitionMessage>createCreditCard</TransitionMessage>
        <Direction>OUT</Direction>
        <TemporalConstraint>NO</TemporalConstraint>
      </Transition>
    </Transitions>
  </TemporalInteractionContract>
</Component>
```
(Continued) TIC of the 'Environment':

```xml
<Transition>
  <SourceState>TransactionRequest</SourceState>
  <DestinationState>TransactionActive</DestinationState>
  <TransitionMessage>startTransaction</TransitionMessage>
  <Direction>OUT</Direction>
  <TemporalConstraint>NO</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>TransactionRequest</SourceState>
  <DestinationState>ShoppingDone</DestinationState>
  <TransitionMessage>shoppingPeriodExpires</TransitionMessage>
  <Direction>NO</Direction>
  <TemporalConstraint>40</TemporalConstraint>
</Transition>

<Transition>
  <SourceState>TransactionActive</SourceState>
  <DestinationState>ShoppingDone</DestinationState>
  <TransitionMessage>shoppingPeriodExpires</TransitionMessage>
  <Direction>NO</Direction>
  <TemporalConstraint>40</TemporalConstraint>
</Transition>

</Transitions>
</Component>
```
Appendix B. Source Code

Initialize.java:

```java
/*
 * Calculate InitialTimeSlice
 * @author: Omkar Jayant Tilak
 * @date: Aug 2006
 */

package code;

import java.util.StringTokenizer;

public class Initialize {

    private static String[][] InitialTimeSlice;
    private static TemporalInteractionContract tic;
    private static String[] States;
    private static String[] Messages;
    private static String[][][] Transitions;

    Initialize() {
        InitialTimeSlice = null;
        tic = null;
        States = null;
        Messages = null;
        Transitions = null;
    }

    static void print1(String a[]) {
        System.out.println("----------------");
        for (int i = 0; i < a.length; i++)
            System.out.println(a[i]);
        // System.out.println("----------------");
    }

    static int index(String yourArray[], String tobefound) // method to find
    // index of
    // specified element
    {
        final int notfound = -1;
        int counter;
        for (counter = 0; counter < yourArray.length; counter++) {
            if (tobefound.equals(yourArray[counter])) // check if we found the
                // number we are looking
                // for
                {
                    return counter;
                }
        } // end for loop
        // if we got this far we must not have found the number
        return notfound;
    }
```
private String getDuration(String state, String message) {
    String duration = null;
    for (int i = 0; i < States.length; i++) {
        if (Transitions[index(States, state)][i][index(Messages, message)] != null) {
            StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][i][index(Messages, message)], "$");
            while (st.hasMoreTokens()) {
                duration = st.nextToken();
            }
        }
    }
    return duration;
}

private static boolean isTransition(String source, String dest, String message) {
    boolean result = false;
    if (Transitions[index(States, source)][index(States, dest)][index(Messages, message)] != null) {
        result = true;
    }
    return result;
}

private static String spawn(String state) {
    String duration = null;
    String result = null;
    for (int j = 0; j < States.length; j++)
        for (int k = 0; k < Messages.length; k++) {
            if ((Transitions[index(States, state)][j][k] != null)) {
                StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][j][k], "$");
                while (st.hasMoreTokens()) {
                    duration = st.nextToken();
                }
            }
        }
    return result;
}
(Continued) Initialize.java:

```java
if (duration.equals("NO")) {
    if (result == null)
        result = Messages[k];
    else
        result = result + "$" + Messages[k];
}
return result;
}

private void getTICData(String filename) {
    tic = new TemporalInteractionContract();
    tic = tic.startParsing(filename);
    States = tic.getStates();
    // System.out.println("In init");
    // print1(States);
    Messages = tic.getMessages();
    Transitions = tic.getTransitions();
    InitialTimeSlice = new String[States.length][Messages.length];
}

private void initializationProcess() {
    int i = 0;
    int j = 0;
    int k = 0;
    boolean[] has_temporal = new boolean[States.length];
    String duration = null;
    String spawn = null;
    String spawn_message = null;
    String temporal_message = null;
    String spawn_state = null;
    StringTokenizer st;
    for (i = 0; i < States.length; i++)
        has_temporal[i] = false;
    for (i = 0; i < States.length; i++)
        for (j = 0; j < States.length; j++)
            for (k = 0; k < Messages.length; k++) {
                if (isTransition(States[i], States[j], Messages[k])) {
                    duration = getDuration(States[i],
```

(Continued) Initialize.java:

```java
if (duration.equals("NO"))
    has_temporal[i] = has_temporal[i] || false;
else
    has_temporal[i] = has_temporal[i] || true;
}
}
for (i = 0; i < States.length; i++) {
    if (has_temporal[i] == true) {
        for (j = 0; j < States.length; j++)
            for (k = 0; k < Messages.length; k++)
                if (isTransition(States[i], States[j],
                    Messages[k])) {
                    duration = getDuration(States[i],
                        Messages[k],
                        Messages[k]);
                    if (!duration.equals("NO")) {
                        temporal_message =
                            // else non_temporal_state =
                            States[j];
                    } else
                        non_temporal_message =
                            States[j];
                    duration = getDuration(States[i],
                        temporal_message);
                    spawn = spawn(States[i]);
                    if (spawn != null) {
                        st = new StringTokenizer(spawn, "$");
                        while (st.hasMoreTokens()) {
                            spawn_message = st.nextToken();
                            for (j = 0; j < States.length; j++)
                                if (isTransition(States[i], States[j],
                                    spawn_message))
                                    spawn_state = States[j];
                    }
                    InitialTimeSlice[i][index(Messages,
                        spawn_message)] = "[0,
                        + duration + "]$" +
                        spawn_state;
                }
            }
        }
    }
}
```
else {
    duration = "INFINITY";
    spawn = spawn(States[i]);
    if (spawn != null) {
        st = new StringTokenizer(spawn, "$");
        while (st.hasMoreTokens()) {
            spawn_message = st.nextToken();
            for (j = 0; j < States.length; j++) {
                if (isTransition(States[i], States[j],
                spawn_message))

                InitialTimeSlice[i][index(Messages,
                spawn_message)] = "[0," + duration
                        + "]$" +
                States[j];
            }
        }
    }
}

public void doInitialize(String filename) {
    Initialize i = new Initialize();
    i.getTICData(filename);
    i.initializationProcess();
}

public String[][] getInitialTimeSlice() {
    return InitialTimeSlice;
}

public void setInitialTimeSlice(String[][] array) {
    InitialTimeSlice = array;
}

public TemporalInteractionContract getTemporalInteractionContract() {
    return tic;
}

public void setTemporalInteractionContract(TemporalInteractionContract t) {
    tic = t;
}
package code;

import java.util.StringTokenizer;

public class TimeSlice {

    private TemporalInteractionContract tic;
    private Initialize init;
    private static String[] States;
    private static String[] Messages;
    private static String[][][] Transitions;
    private static String[] InitialTimeSlice;
    private static String[] TimeSlice;
    private static String[][] InitialTimeSlice;
    private static String StateSet;
    private static String[] TimeSlice;

    TimeSlice() {
        States = null;
        Messages = null;
        Transitions = null;
        InitialTimeSlice = null;
        StateSet = null;
        TimeSlice = null;
        tic = null;
        init = null;
    }

    private static String getDuration(String state, String message) {
        String duration = null;
        for (int i = 0; i < States.length; i++) {
            if (Transitions[index(States, state)][i][index(Messages, message)] != null) {
                StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][i][index(Messages, message)], "\$");
                while (st.hasMoreTokens()) {
                    duration = st.nextToken();
                }
            }
        }
        return duration;
    }
}
(Continued) TimeSlice.java:

```java
private static int index(String yourArray[], String tobefound) // method to
// find
// index of
// specified
// element
{
    final int notfound = -1;
    int counter;
    for (counter = 0; counter < yourArray.length; counter++) {
        if (tobefound.equals(yourArray[counter])) // check if we found the
            // number we are looking
                // for
        {
            return counter;
        }
    } // end for loop
    // if we got this far we must not have found the number
    return notfound;
}

private static boolean isTransition(String source, String dest, String message) {
    boolean result = false;
    if (Transitions[index(States, source)][index(States, dest)][index(
            Messages, message)] != null) {
        result = true;
    }
    return result;
}

private static float min(float a, float b) {
    if (a < b)
        return a;
    else
        return b;
}

private static float max(float a, float b) {
    if (a > b)
        return a;
    else
        return b;
}
```
(Continued) TimeSlice.java:

```java
private static float toFloat(String s) {
    float f = (float) 0;
    if (s == null)
        return f;
    if (s.equals("INFINITY"))
        f = Float.POSITIVE_INFINITY;
    else if (s != null)
        f = Float.parseFloat(s);
    return f;
}

private static String toString(float f) {
    return Float.toString(f);
}

private static float[] getFloatTimeValues(String s) {
    String temp = null, part1, part2;
    int index;
    float[] result = new float[2];
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "$);
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (temp.startsWith("[") {
                index = temp.indexOf(",");
                part1 = temp.substring(1, index);
                part2 = temp.substring(index + 1, temp.length() - 1);
                result[0] = toFloat(part1);
                result[1] = toFloat(part2);
            }
        }
    }
    return result;
}

private static String getState(String s) {
    String temp = null, result = null;
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "$");  
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (!temp.startsWith("[")) {
                result = temp;
            }
        }
    }
    return result;
}
```
(Continued) TimeSlice.java:

```java
private static String spawn(String state) {
    String duration = null;
    String result = null;

    for (int j = 0; j < States.length; j++)
        for (int k = 0; k < Messages.length; k++) {
            if ((Transitions[index(States, state)][j][k] != null)) {
                StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][j][k], "$");
                while (st.hasMoreTokens()) {
                    duration = st.nextToken();
                }

                if (duration.equals("NO")) {
                    if (result == null)
                        result = Messages[k];
                    else
                        result = result + "$" + Messages[k];
                }
            }
        }
    return result;
}

private static boolean normalize_Overlap(String a, String b) {
    boolean result = false;
    float[] result1 = new float[2];
    float[] result2 = new float[2];

    result1 = getFloatTimeValues(a);
    result2 = getFloatTimeValues(b);
    if (((result2[0] <= result1[0]) && (result1[0] <= result2[1]))
        || ((result2[0] <= result1[1]) && (result1[1] <= result2[1])))
        result = true;
    return result;
}
```
(Continued) TimeSlice.java:

```java
private static String add(String s, String add) {
    String result = null, temp, state;
    float[] slice = new float[2];
    float x;
    x = toFloat(add);
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "|");
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            slice = getFloatTimeValues(temp);
            state = getState(temp);
            slice[0] = slice[0] + x;
            temp = "[" + toString(slice[0]) + "," + toString(slice[1])
                                + "]$" + state;
            if (result == null)
                result = temp;
            else
                result = result + "|" + temp;
        }
    }
    return result;
}

private static String normalize(String s) {
    String result = null, temp, istate, jstate;
    String[] elements = null;
    int i = 0, j = 0;
    float[] slice1 = new float[2];
    float[] slice2 = new float[2];
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "|");
        elements = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            elements[i] = temp;
            i++;
        }
    }
}
(Continued) TimeSlice.java:

```java
for (i = 0; i < elements.length; i++) {
    if (elements[i] != null) {
        istate = getState(elements[i]);
        for (j = 0; j < elements.length; j++) {
            if ((j != i) && (elements[j] != null)) {
                jstate = getState(elements[j]);
                if (istate.equals(jstate)) {
                    if (normalize_Overlap(elements[i], elements[j])) {
                        slice1 = getFloatTimeValues(elements[i]);
                        slice2 = getFloatTimeValues(elements[j]);
                        elements[i] = "[" + toString(min(slice1[0], slice2[0])) + ","
                                      + toString(max(slice1[1], slice2[1])) + "]" + istate;
                        elements[j] = null;
                    }
                }
            }
        }
    } // if elements[i]
} // for i loop

for (i = 0; i < elements.length; i++) {
    if (elements[i] != null) {
        if (result == null)
            result = elements[i];
        else
            result = result + "|" + elements[i];
    }
} 

result = result + "|" + elements[i];
```

return result;
```
private static String union(String a, String b) {
    String result = null;
    if ((a == null) && (b != null))
        result = b;
    else if ((a != null) && (b == null))
        result = a;
    else if ((a != null) && (b != null))
        result = a + "|" + b;
    return result;
}

private void doSetup(String filename) {
    tic = new TemporalInteractionContract();
    init = new Initialize();
    init.doInitialize(filename);
    tic = init.getTemporalInteractionContract();
    States = tic.getStates();
    Messages = tic.getMessages();
    Transitions = tic.getTransitions();
    InitialTimeSlice = init.getInitialTimeSlice();
    TimeSlice = new String[Messages.length];
}

private static void calculateTimeSlice(String sinput) {
    int i = 0;
    int j = 0;
    int k = 0;
    boolean[] has_temporal = new boolean[States.length];
    String duration = null;
    String LastState = sinput;
    String temporal_state = null;
    String temporal_message = null;
    String spawn = null;
    String spawn_message = null;
    float TimeElapsed = 0;
    String Temp = "$";
    StateSet = sinput;
    // TimeSlice = new String[Messages.length];
for (i = 0; i < States.length; i++)
    has_temporal[i] = false;
for (i = 0; i < States.length; i++)
    for (j = 0; j < States.length; j++)
        for (k = 0; k < Messages.length; k++) {
            if (isTransition(States[i], States[j], Messages[k])) {
                duration = getDuration(States[i], Messages[k]);
                if (duration.equals("NO"))
                    has_temporal[i] = has_temporal[i] ||
                        false;
                else
                    has_temporal[i] = has_temporal[i] ||
                        true;
            }
        }
while (!(Temp.equals(StateSet))) {
    Temp = StateSet;
    if (has_temporal[index(States, LastState)] == true) {
        for (i = 0; i < States.length; i++)
            for (j = 0; j < Messages.length; j++) {
                if (isTransition(LastState, States[i], Messages[j])) {
                    duration = getDuration(LastState, Messages[j]);
                    if (!duration.equals("NO")) {
                        temporal_state = States[i];
                        temporal_message =
                            Messages[j];
                        // TimeElapsed = TimeElapsed +
                        // toFloat(getDuration(LastState,temporal_message));
                        StateSet = union(StateSet, temporal_state);
                        spawn = spawn(LastState);
                        // LastState = temporal_state;
                    }
                }
            }
    }
    Temp = StateSet;
}
if (spawn != null) { StringTokenizer st = new StringTokenizer(spawn, "$"); while (st.hasMoreTokens()) {
    spawn_message = st.nextToken();
    TimeSlice[index(Messages, spawn_message)] = normalize(union(
        InitialTimeSlice[index(States, LastState)][index(Messages, spawn_message)],
        toString(TimeElapsed)));
    } TimeElapsed = TimeElapsed + toFloat(getDuration(LastState, temporal_message));
    LastState = temporal_state;
} // if has_temporal
else {
    spawn = spawn(LastState);
    if (spawn != null) {
        StringTokenizer st = new StringTokenizer(spawn, "$"); while (st.hasMoreTokens()) {
            spawn_message = st.nextToken();
            TimeSlice[index(Messages, spawn_message)] = normalize(union(
                InitialTimeSlice[index(States, LastState)][index(Messages, spawn_message)],
                toString(TimeElapsed)));
        } // end else
    } // end while
} // end while
(Continued) TimeSlice.java:

```java
public void startProcessing(String filename) {
    TimeSlice t = new TimeSlice();
    t.doSetup(filename);
}

public String[] getTimeSlice(String state) {
    for (int i = 0; i < Messages.length; i++)
        TimeSlice[i] = null;
    calculateTimeSlice(state);
    return TimeSlice;
}

public void setTimeSlice(String[] array) {
    TimeSlice = array;
}

public String getStateSet() {
    return StateSet;
}

public void setStateSet(String s) {
    StateSet = s;
}
```
TemporalInteractionContract.java:

```java
/*
 * Generate TIC elements from XML file
 * @author: Omkar Jayant Tilak
 * @date:Aug 2006
 * */

package code;

import java.io.IOException;
import java.util.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class TemporalInteractionContract {
    private String ComponentName;
    private String[] States;
    private String[] StartState;
    private String[] EndStates;
    private String[] Messages;
    private String[] Tran;
    private String[][][] Transitions;
    private String type;
    private int statecount;
    private int statectr;
    private int startstatecount;
    private int startstatectr;
    private int endstatecount;
    private int endstatectr;
    private int messagecount;
    private int messagectr;
    private int transitioncount;
    private int transitionctr;
}
```
TemporalInteractionContract() {  
ComponentName = null;  
States = null;  
StartState = null;  
EndStates = null;  
Messages = null;  
Tran = null;  
Transitions = null;  
type = "";  
statecount = 0;  
statectr = 0;  
startstatecount = 0;  
startstatectr = 0;  
endstatecount = 0;  
endstatectr = 0;  
messagecount = 0;  
messagectr = 0;  
transitioncount = 0;  
transitionctr = 0;  
}

static int index(String yourArray[], String tobefound) // method to find  
  // index of  
  // specified element  
{
  final int notfound = -1;  
  int counter;
  for (counter = 0; counter < yourArray.length; counter++) {
    if (tobefound.equals(yourArray[counter])) // check if we found the  
      // number we are looking  
      // for  
      {
        return counter;
      }
  } // end for loop  
  // if we got this far we must not have found the number  
  return notfound;
}

private void assignArray() {  
States = new String[statecount];  
StartState = new String[startstatecount];  
EndStates = new String[endstatecount];  
Messages = new String[messagecount];  
Tran = new String[transitioncount];  
Transitions = new String[statecount][statecount][messagecount];
}
private void firstparse(Node node) throws IOException {
    NodeList nodes = null;
    if (node.getNodeType() == Node.DOCUMENT_NODE) {
        nodes = node.getChildNodes();
        if (nodes != null) {
            for (int i = 0; i < nodes.getLength(); i++) {
                firstparse(nodes.item(i));
            }
        }
    } else if (node.getNodeType() == Node.ELEMENT_NODE) {
        if (node.getNodeName().equals("ComponentName")) {
            type = "ComponentName";
        } else if (node.getNodeName().equals("StartState")) {
            startstatecount = startstatecount + 1;
            statecount = statecount + 1;
        } else if (node.getNodeName().equals("EndState")) {
            endstatecount = endstatecount + 1;
            statecount = statecount + 1;
        } else if (node.getNodeName().equals("State")) {
            statecount = statecount + 1;
        } else if (node.getNodeName().equals("Message")) {
            messagecount = messagecount + 1;
        } else if (node.getNodeName().equals("Transition")) {
            transitioncount = transitioncount + 1;
        }
        nodes = node.getChildNodes();
        if (nodes != null) {
            for (int i = 0; i < nodes.getLength(); i++) {
                firstparse(nodes.item(i));
            }
        }
    }
}

private void secondparse(Node node) throws IOException {
    String temp = "";
    NodeList nodes = null;
    if (node.getNodeType() == Node.DOCUMENT_NODE) {
        nodes = node.getChildNodes();
        if (nodes != null) {
            for (int i = 0; i < nodes.getLength(); i++) {
                secondparse(nodes.item(i));
            }
        }
    }
}
else if (node.getNodeType() == Node.ELEMENT_NODE) {
    if (node.getNodeName().equals("ComponentName")) {
        type = "ComponentName";
    } else if (node.getNodeName().equals("StartState")) {
        type = "StartState";
    } else if (node.getNodeName().equals("EndState")) {
        type = "EndState";
    } else if (node.getNodeName().equals("State")) {
        type = "State";
    } else if (node.getNodeName().equals("Message")) {
        type = "Message";
    } else if (node.getNodeName().equals("SourceState")
             || node.getNodeName().equals("TransitionMessage")
             || node.getNodeName().equals("DestinationState")
             || node.getNodeName().equals("Direction")) {
        type = "Transition";
    } else if (node.getNodeName().equals("TemporalConstraint")) {
        type = "Temporal";
    }
    nodes = node.getChildNodes();
    if (nodes != null) {
        for (int i = 0; i < nodes.getLength(); i++) {
            secondparse(nodes.item(i));
        }
    }
}
else if (node.getNodeType() == Node.TEXT_NODE) {
    temp = node.getNodeValue();
    if (type.equals("ComponentName")) {
        ComponentName = temp;
    }
    if (type.equals("StartState")) {
        States[statectr] = new String(temp);
        statectr = statectr + 1;
        StartState[startstatectr] = new String(temp);
        startstatectr = startstatectr + 1;
    }
    if (type.equals("EndState")) {
        States[statectr] = new String(temp);
        statectr = statectr + 1;
        EndStates[endstatectr] = new String(temp);
        endstatectr = endstatectr + 1;
    }
}
(Continued) TemporalInteractionContract.java:

```java
if (type.equals("State")) {
    States[statectr] = new String(temp);
    statectr = statectr + 1;
}
if (type.equals("Message")) {
    Messages[messagectr] = new String(temp);
    messagectr++;
}
if (type.equals("Transition")) {
    if (Tran[transitionctr] != null)
        Tran[transitionctr] = Tran[transitionctr] + temp + "$";
    if (Tran[transitionctr] == null)
        Tran[transitionctr] = temp + "$";
}
if (type.equals("Temporal")) {
    Tran[transitionctr] = Tran[transitionctr] + temp;
    transitionctr++;
}
type = "";
}

private void thirdparse() {
    for (int i = 0; i < Tran.length; i++) {
        if (Tran[i] != null) {
            StringTokenizer st = new StringTokenizer(Tran[i], "$");
            while (st.hasMoreTokens()) {
                // System.out.println(st.nextToken());
                Transitions[index(States, st.nextToken())][index(States, st.nextToken())][index(Messages, st.nextToken())] = st.nextToken() + "$" + st.nextToken();
            }
        }
    }
}
```
public TemporalInteractionContract startParsing(String filename) {
    TemporalInteractionContract dom = new TemporalInteractionContract();
    Document doc;
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        doc = builder.parse(filename);
        dom.firstparse(doc);
        dom.assignArray();
        dom.secondparse(doc);
        dom.thirdparse();
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    } catch (SAXException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return (dom);
}

public String getComponentName() {
    return ComponentName;
}

public void setComponentName(String name) {
    ComponentName = name;
}

public String[] getStates() {
    return States;
}

public void setStates(String[] array) {
    States = array;
}

public String[][][][] getTransitions() {
    return Transitions;
}

public void setTransitions(String[][][] array) {
    Transitions = array;
}

public String[] getStartState() {
    return StartState;
}

public void setStartState(String[] array) {
    StartState = array;
}

public String[] getEndStates() {
    return EndStates;
}

public void setEndStates(String[] array) {
    EndStates = array;
}
/*
 * Calculate InteractionTrace of two TICs
 * @author: Omkar Jayant Tilak
 * @date: Aug 2006
 * */

package code;

import java.util.StringTokenizer;

public class InteractionTrace {

    private TemporalInteractionContract tic1;
    private TemporalInteractionContract tic2;
    private static TimeSlice ts1;
    private Initialize init1;
    private Initialize init2;
    private static String[] States1;
    private static String[] StartState1;
    private static String[] EndStates1;
    private static String[] States2;
    private static String[] StartState2;
    private static String[] EndStates2;
    private static String[] Messages1;
    private static String[] Messages2;
    private static String[][] Transitions1;
    private static String[][] Transitions2;
    private static String[][] TimeSlice1;
    private static String[][] TimeSlice2;
    private static String[] StateSet1;
    private static String[] StateSet2;
    private static String[][][] InteractionTimeSlice;
    private static String Combined;
    private static String Initial;
    private static String StartSet;
    private static String FinishSet;

}
(Continued) InteractionTrace.java:

```java
InteractionTrace() {
    tic1 = null;
    tic2 = null;
    ts1 = null;
    init1 = null;
    init2 = null;
    States1 = null;
    StartState1 = null;
    EndStates1 = null;
    States2 = null;
    StartState2 = null;
    EndStates2 = null;
    Messages1 = null;
    Messages2 = null;
    Transitions1 = null;
    Transitions2 = null;
    TimeSlice1 = null;
    TimeSlice2 = null;
    StateSet1 = null;
    StateSet2 = null;
    InteractionTimeSlice = null;
    Combined = null;
    Initial = null;
    StartSet = null;
    FinishSet = null;
}

static void print32(String[][][] array) {
    int x = array.length;
    int y = array[0].length;
    int z = array[0][0].length;
    String[] MaxMessages = null;
    if (Messages1.length > Messages2.length)
        MaxMessages = Messages1;
    else
        MaxMessages = Messages2;
    // System.out.println(x +"+y+" +z);
    for (int i = 0; i < x; i++) {
        for (int j = 0; j < y; j++) {
            for (int k = 0; k < z; k++) {
                if(array[i][j][k] != null)
                {
                }
            }
        }
    }
```

private static float min(float a, float b) {
    if (a < b)
        return a;
    else
        return b;
}

private static float max(float a, float b) {
    if (a > b)
        return a;
    else
        return b;
}

private static float toFloat(String s) {
    float f = (float) 0;
    if (s == null)
        return f;
    if (s.equals("INFINITY"))
        f = Float.POSITIVE_INFINITY;
    else if (s != null)
        f = Float.parseFloat(s);
    return f;
}

private static String getState(String s) {
    String temp = null, result = null;
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "$");
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (!(temp.startsWith("["))) {
                result = temp;
            }
        }
    }
    return result;
}
private static float[] getTime(String s) {
    String temp = null, part1, part2;
    int index;
    float[] result = new float[2];
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "\$");
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (temp.startsWith("[
                        ")) {
                index = temp.indexOf(",
                        ");
                part1 = temp.substring(1, index);
                part2 = temp.substring(index + 1, temp.length() - 1);
                result[0] = toFloat(part1);
                result[1] = toFloat(part2);
            }
        }
    }
    return result;
}

private static String getDirection(String state, String message,
    String[] States, String[] Messages, String[][][] Transitions) { 
    String direction = null;
    for (int i = 0; i < States.length; i++) {
        if (Transitions[index(States, state)][i][index(Messages, message)] != null) {
            StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][i][index(Messages, message)], 
                        "\$");
            while (st.hasMoreTokens()) { 
                direction = st.nextToken();
                st.nextToken();
            }
        }
    }
    return direction;
}
private static int index(String yourArray[], String tobefound)// method to
// find
// index of
// specified
// element
{
    final int notfound = -1;
    int counter;
    for (counter = 0; counter < yourArray.length; counter++) {
        if (tobefound.equals(yourArray[counter]))// check if we found the
            // number we are looking
                // for
        {
            return counter;
        }
    }// end for loop
    // if we got this far we must not have found the number
    return notfound;
}

private static String union(String a, String b) {
    String result = null;
    if ((a == null) && (b != null))
        result = b;
    else if ((a != null) && (b == null))
        result = a;
    else if ((a != null) && (b != null))
        result = a + "|" + b;
    return result;
}

static boolean intersect(String a, String[] b) {
    boolean result = false;
    String[] temp = null;
    int i = 0, j = 0;
    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, "|");
        temp = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            temp[i] = st.nextToken();
            i++;
        }
    }
(Continued) InteractionTrace.java:

```java
for (i = 0; i < temp.length; i++)
    for (j = 0; j < b.length; j++)
        if (temp[i].equals(b[j])) {
            result = true;
            break;
        }
return result;
}

static boolean similar(String a, String b) {
    boolean result = false;
    String[] astring = new String[2];
    String[] bstring = new String[2];
    int i = 0;

    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, "\$");
        while (st.hasMoreTokens()) {
            astring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;

    if (b != null) {
        StringTokenizer st = new StringTokenizer(b, "\$");
        while (st.hasMoreTokens()) {
            bstring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;

    if ((a != null) && (b != null)) {
        if ((astring[0].equals(bstring[0]))
            && (astring[1].equals(bstring[1])))
            result = true;
        if ((astring[0].equals(bstring[1]))
            && (astring[1].equals(bstring[0])))
            result = true;
    }
    if ((a == null) && (b == null))
        result = false;
    return result;
```
static String subtract(String a) {
    String result = null;
    String[] astring = null;
    int i = 0;
    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, "|");
        astring = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            astring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;
    for (i = 1; i < astring.length; i++) {
        if (result == null) result = astring[i];
        else result = result + "|" + astring[i];
    }
    return result;
}
private static String spawn(String state, String[] States, 
    String[] Messages, String[][] Transitions) {
    String direction = null;
    String duration = null;
    String result = null;
    for (int j = 0; j < States.length; j++)
        for (int k = 0; k < Messages.length; k++) {
            if ((Transitions[index(States, state)][j][k] != null)) {
                StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][j][k], "$");
                while (st.hasMoreTokens()) {
                    duration = st.nextToken();
                    direction = st.nextToken();
                }
                if (duration.equals("NO")) {
                    if (result == null)
                        result = Messages[k];
                    else
                        result = result + "$" + Messages[k];
                }
            }
        }
    return result;
}
(Continued) InteractionTrace.java:

```java
void doInitialSetup(String filename1, String filename2) {
    String[] temp = null;
    int i, j, maxstates, maxmessages;

    //tic1 = new TemporalInteractionContract();
    tic2 = new TemporalInteractionContract();
    ts1 = new TimeSlice();
    init1 = new Initialize();
    init1.doInitialize(filename1);
    tic1 = init1.getTemporalInteractionContract();
    States1 = tic1.getStates();
    //System.out.println("In trace");
    //print1(States1);
    StartState1 = tic1.getStartState();
    EndStates1 = tic1.getEndStates();
    Messages1 = tic1.getMessages();
    Transitions1 = tic1.getTransitions();
    TimeSlice1 = new String[States1.length][Messages1.length];
    StateSet1 = new String[States1.length];

    init2 = new Initialize();
    init2.doInitialize(filename2);
    tic2 = init2.getTemporalInteractionContract();
    StartState2 = tic2.getStartState();
    EndStates2 = tic2.getEndStates();
    States2 = tic2.getStates();
    Messages2 = tic2.getMessages();
    Transitions2 = tic2.getTransitions();
    TimeSlice2 = new String[States2.length][Messages2.length];
    StateSet2 = new String[States2.length];

    if (States1.length > States2.length)
        maxstates = States1.length;
    else
        maxstates = States2.length;
    if (Messages1.length > Messages2.length)
        maxmessages = Messages1.length;
    else
        maxmessages = Messages2.length;
    InteractionTimeSlice = new String[maxstates][maxstates][maxmessages];
}
```
(Continued) InteractionTrace.java:

ts1.startProcessing(filename1);
    temp = new String[Messages1.length];
    for (i = 0; i < States1.length; i++) {
        temp = ts1.getTimeSlice(States1[i]);
        for (j = 0; j < temp.length; j++)
            TimeSlice1[i][j] = temp[j];
        StateSet1[i] = ts1.getStateSet();
    }
    // print1(StateSet1);
    ts1.startProcessing(filename2);
    temp = new String[Messages2.length];
    for (i = 0; i < States2.length; i++) {
        temp = ts1.getTimeSlice(States2[i]);
        for (j = 0; j < temp.length; j++)
            TimeSlice2[i][j] = temp[j];
        StateSet2[i] = ts1.getStateSet();
    }
    // print1(StateSet2);
}

void calculateIntersection() {
    String temp1, temp2, tuple, spawn, state1, state2, dir1, dir2;
    String[] spawn_messages1 = null;
    String[] spawn_messages2 = null;
    String[] array1;
    String[] array2;
    String element1 = null, element2 = null;
    StringTokenizer st, st1;
    int i = 0, j = 0, ctr = 0, iin = 0, jin = 0;
    float minimum, maximum;
    float[] slice1 = new float[2];
    float[] slice2 = new float[2];
    Combined = StartState1[0] + "$" + StartState2[0];
    Initial = StartState1[0] + "$" + StartState2[0];
    boolean falseAlarm = false;
    while (Combined != null) {
        st = new StringTokenizer(Combined, "|");
        tuple = st.nextToken();
        st = new StringTokenizer(tuple, "$");
        temp1 = st.nextToken();
        temp2 = st.nextToken();
        StartSet = union(StartSet, temp1 + "$" + temp2);
        Combined = subtract(Combined);
        spawn = spawn(temp1, States1, Messages1, Transitions1);
if (spawn != null) {
    st = new StringTokenizer(spawn, "$");
    spawn_messages1 = new String[st.countTokens()];
    while (st.hasMoreTokens()) {
        spawn_messages1[i] = st.nextToken();
        i++;
    }
    i = 0;
}

spawn = spawn(temp2, States2, Messages2, Transitions2);
if (spawn != null) {
    st = new StringTokenizer(spawn, "$");
    spawn_messages2 = new String[st.countTokens()];
    while (st.hasMoreTokens()) {
        spawn_messages2[i] = st.nextToken();
        i++;
    }
    i = 0;
}

for (i = 0; i < spawn_messages1.length; i++)
    for (j = 0; j < spawn_messages2.length; j++) {
        dir1 = getDirection(temp1, spawn_messages1[i],
                               States1, Messages1, Transitions1);
        dir2 = getDirection(temp2, spawn_messages2[j],
                               States2, Messages2, Transitions2);
        if ((dir1 != null) && (dir2 != null)) {
            if ((dir1.equals("IN") && dir2.equals("OUT"))) {
                element1 =
                TimeSlice1[index(States1, temp1)][index(
                    spawn_messages1[i])];
                element2 =
                TimeSlice2[index(States2, temp2)][index(
                    spawn_messages2[j])];

                st = new StringTokenizer(element1,
                                          "|");
                array1 = new String[st.countTokens()];
                while (st.hasMoreTokens()) {
                    array1[ctr] = st.nextToken();
                    ctr++;
                }
            }
        }
    }
(Continued) InteractionTrace.java:

```java
ctr = 0;
"|");
String[st.countTokens()];

array2.length; jin++) {
getTime(array1[jin]);
getTime(array2[jin]);
getState(array1[jin]);
getState(array2[jin]);
slice2[0] <= min(
    slice1[1], slice2[1])) {
    min(slice1[1], slice2[1]);
    max(slice1[0], slice2[0]);
    != maximum)
```

```java
array2 = new String[st.countTokens()];
while (st.hasMoreTokens()) {
    array2[ctr] = st.nextToken();
    ctr++;
}
```

```java
ctr = 0;
for (iin = 0; iin < array1.length; iin++)
for (jin = 0; jin <
slice1 =
slice2 =
state1 =
state2 =
if (max(slice1[0],
```
(Continued) InteractionTrace.java:

```java
InteractionTimeSlice[index(States1,
    temp1)][index(States2, temp2)][index(
    Messages1, spawn_messages1[i])]) = union(
    InteractionTimeSlice[index(
        States1, temp1)][index(
        States2, temp2)][index(
        Messages1, spawn_messages1[i])],
    "[" + maximum + "," + minimum
    + "]$" + state1 + "$"
    + state2);

    //System.out.println(state1+state2);
    Combined = union(Combined, state1 + "$"
    + state2);

    } //end if min
    max

    } // if spawn ends
    } // if dir null check ends
    } // outermost for ends

    i = 0;
    j = 0;

    } // while ends

    st = new StringTokenizer(StartSet, "|");
    while (st.hasMoreTokens()) {
        tuple = st.nextToken();
        st1 = new StringTokenizer(tuple, "$");
        temp1 = st1.nextToken();
        temp2 = st1.nextToken();
```

```
(Continued) InteractionTrace.java:

```java
if (intersect(StateSet1[index(States1, temp1)], EndStates1)
    && intersect(StateSet2[index(States2, temp2)],
                  EndStates2)) {
    for(i=0;i<InteractionTimeSlice[0][0].length;i++)
        {  
            if(InteractionTimeSlice[index(States1,temp1)][index(States2,temp2)][i] != null)
                falseAlarm = true;
        }
    if(falseAlarm == false)
        {  
            FinishSet = union(FinishSet, temp1 + "$" + temp2);
        }
    falseAlarm = false;
}  //end of if
}  //end of while tokens

public void beginCalculation(String filename1, String filename2) {
    InteractionTrace it = new InteractionTrace();
    it.doInitialSetup(filename1, filename2);
    it.calculateIntersection();
}

public void printInteractionTrace()
    {print32(this.InteractionTimeSlice);}

public String getStartSet() {
    return StartSet;
}

public void setStartSet(String s) {
    StartSet = s;
}

public String getInitial() {
    return Initial;
}

public void setInitial(String s) {
    Initial = s;
}

public String getFinishSet() {
    return FinishSet;
}

public void setFinishSet(String s) {
    FinishSet = s;
}

public String[][][] getIntersectionTimeSlice() {
    return InteractionTimeSlice;
}

public void setIntersectionTimeSlice(String[][][] array) {
    InteractionTimeSlice = array;
}

public String[][] getTimeSlice1()
    {return TimeSlice1;}

public String[][] getTimeSlice2() {
    return TimeSlice2;
}
```
/**
 * Calculate InteractionTrace of two TICs
 * @author Omkar Jayant Tilak
 * @date Aug 2006
 **/

package code;

import java.util.StringTokenizer;

public class InteractionTrace {

    private TemporalInteractionContract tic1;
    private TemporalInteractionContract tic2;
    private static TimeSlice ts1;
    private Initialize init1;
    private Initialize init2;
    private static String[] States1;
    private static String[] StartState1;
    private static String[] EndStates1;
    private static String[] States2;
    private static String[] StartState2;
    private static String[] EndStates2;
    private static String[] Messages1;
    private static String[] Messages2;
    private static String[][][] Transitions1;
    private static String[][][] Transitions2;
    private static String[][] TimeSlice1;
    private static String[][] TimeSlice2;
    private static String[] StateSet1;
    private static String[] StateSet2;
    private static String[][][] IntersectionTimeSlice;
    private static String[] Combined;
    private static String Initial;
    private static String StartSet;
    private static String FinishSet;
}
(Continued) Intersection.java:

```java
InteractionTrace() {
    tic1 = null;
    tic2 = null;
    ts1 = null;
    init1 = null;
    init2 = null;
    States1 = null;
    StartState1 = null;
    EndStates1 = null;
    States2 = null;
    StartState2 = null;
    EndStates2 = null;
    Messages1 = null;
    Messages2 = null;
    Transitions1 = null;
    Transitions2 = null;
    TimeSlice1 = null;
    TimeSlice2 = null;
    StateSet1 = null;
    StateSet2 = null;
    InteractionTimeSlice = null;
    Combined = null;
    Initial = null;
    StartSet = null;
    FinishSet = null;
}

static void print32(String[][][] array) {
    int x = array.length;
    int y = array[0].length;
    int z = array[0][0].length;
    String[] MaxMessages = null;
    if (Messages1.length > Messages2.length)
        MaxMessages = Messages1;
    else
        MaxMessages = Messages2;
    // System.out.println(x+" \+y+" \+z);
    for (int i = 0; i < x; i++) {
        for (int j = 0; j < y; j++) {
            for (int k = 0; k < z; k++) {
                if (array[i][j][k] != null)
                    System.out.println("InteractionTrace[" + States1[i] + ",
                " + States2[j] + "," + MaxMessages[j] + "] = " + array[i][j][k]);
            }
        }
    }
```

(Continued) Intersection.java:

```java
private static float min(float a, float b) {
    if (a < b)
        return a;
    else
        return b;
}

private static float max(float a, float b) {
    if (a > b)
        return a;
    else
        return b;
}

private static float toFloat(String s) {
    float f = (float) 0;
    if (s == null)
        return f;
    if (s.equals("INFINITY"))
        f = Float.POSITIVE_INFINITY;
    else if (s != null)
        f = Float.parseFloat(s);
    return f;
}

private static String getState(String s) {
    String temp = null, result = null;
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "\$");    
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (!temp.startsWith("\[")) {
                result = temp;
            }
        }
    }
    return result;
}
```
private static float[] getTime(String s) {
    String temp = null, part1, part2;
    int index;
    float[] result = new float[2];
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "$");
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (temp.startsWith("[")) {
                index = temp.indexOf(",");
                part1 = temp.substring(1, index);
                part2 = temp.substring(index + 1, temp.length() - 1);
                result[0] = toFloat(part1);
                result[1] = toFloat(part2);
            }
        }
    }
    return result;
}
private static String getDirection(String state, String message,
    String[] States, String[] Messages, String[][][] Transitions) {
    String direction = null;
    for (int i = 0; i < States.length; i++) {
        if (Transitions[index(States, state)][i][index(Messages, message)] !=
            null) {
            StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][i][index(Messages, message)],
                "$");
            while (st.hasMoreTokens()) {
                direction = st.nextToken();
                st.nextToken();
            }
        }
    }
    return direction;
}
private static int index(String yourArray[], String tobefound) // method to
// find
// index of
// specified
// element
{
    final int notfound = -1;
    int counter;
    for (counter = 0; counter < yourArray.length; counter++) {
        if (tobefound.equals(yourArray[counter])) // check if we found the
            // number we are looking
            // for
            {
            return counter;
        }
    } // end for loop
    // if we got this far we must not have found the number
    return notfound;
}

private static String union(String a, String b) {
    String result = null;
    if ((a == null) && (b != null))
        result = b;
    else if ((a != null) && (b == null))
        result = a;
    else if ((a != null) && (b != null))
        result = a + "|" + b;
    return result;
}

static boolean intersect(String a, String[] b) {
    boolean result = false;
    String[] temp = null;
    int i = 0, j = 0;
    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, "|");
        temp = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            temp[i] = st.nextToken();
            i++;
        }
    }
for (i = 0; i < temp.length; i++)
    for (j = 0; j < b.length; j++)
        if (temp[i].equals(b[j])) {
            result = true;
            break;
        }
return result;
}

static boolean similar(String a, String b) {
    boolean result = false;
    String[] astring = new String[2];
    String[] bstring = new String[2];
    int i = 0;

    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, "$");
        while (st.hasMoreTokens()) {
            astring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;

    if (b != null) {
        StringTokenizer st = new StringTokenizer(b, "$");
        while (st.hasMoreTokens()) {
            bstring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;

    if ((a != null) && (b != null)) {
        if ((astring[0].equals(bstring[0]))
            && (astring[1].equals(bstring[1])))
            result = true;
        if ((astring[0].equals(bstring[1]))
            && (astring[1].equals(bstring[0])))
            result = true;
    }
    if ((a == null) && (b == null))
        result = false;
return result;
}
(Continued) Intersection.java:

```java
static String subtract(String a) {
    String result = null;
    String[] astring = null;
    int i = 0;

    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, "|");
        astring = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            astring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;
    for (i = 1; i < astring.length; i++) {
        if (result == null) result = astring[i];
        else result = result + "|" + astring[i];
    }
    return result;
}

private static String spawn(String state, String[] States, String[] Messages, String[][][] Transitions) {
    String direction = null;
    String duration = null;
    String result = null;

    for (int j = 0; j < States.length; j++)
        for (int k = 0; k < Messages.length; k++) {
            if ((Transitions[index(States, state)][j][k] != null)) {
                StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][j][k], "$");
                while (st.hasMoreTokens()) {
                    direction = st.nextToken();
                    duration = st.nextToken();
                }
                if (duration.equals("NO")) {
                    if (result == null)
                        result = Messages[k];
                    else
                        result = result + "$" + Messages[k];
                }
            }
        }
    return result;
}
```
Continued) Intersection.java:

```java
void doInitialSetup(String filename1, String filename2) {
    String[] temp = null;
    int i, j, maxstates, maxmessages;

    //tic1 = new TemporalInteractionContract();
tic2 = new TemporalInteractionContract();
ts1 = new TimeSlice();

    init1 = new Initialize();
    init1.doInitialize(filename1);
tic1 = init1.getTemporalInteractionContract();
States1 = tic1.getStates();
//System.out.println("In trace");
//print1(States1);
StartState1 = tic1.getStartState();
EndStates1 = tic1.getEndStates();
Messages1 = tic1.getMessages();
Transitions1 = tic1.getTransitions();
TimeSlice1 = new String[States1.length][Messages1.length];
StateSet1 = new String[States1.length];

    init2 = new Initialize();
    init2.doInitialize(filename2);
tic2 = init2.getTemporalInteractionContract();
StartState2 = tic2.getStartState();
EndStates2 = tic2.getEndStates();
States2 = tic2.getStates();
Messages2 = tic2.getMessages();
Transitions2 = tic2.getTransitions();
TimeSlice2 = new String[States2.length][Messages2.length];
StateSet2 = new String[States2.length];

if (States1.length > States2.length)
    maxstates = States1.length;
else
    maxstates = States2.length;
if (Messages1.length > Messages2.length)
    maxmessages = Messages1.length;
else
    maxmessages = Messages2.length;
InteractionTimeSlice = new String[maxstates][maxstates][maxmessages];
```
void calculateIntersection() {
    String temp1, temp2, tuple, spawn, state1, state2, dir1, dir2;
    String[] spawn_messages1 = null;
    String[] spawn_messages2 = null;
    String[] array1;
    String[] array2;
    String element1 = null, element2 = null;
    StringTokenizer st, st1;
    int i = 0, j = 0, ctr = 0, iin = 0, jin = 0;
    float minimum, maximum;
    float[] slice1 = new float[2];
    float[] slice2 = new float[2];
    String Combined = StartState1[0] + "$" + StartState2[0];
    String Initial = StartState1[0] + "$" + StartState2[0];
    boolean falseAlarm = false;
    while (Combined != null) {
        st = new StringTokenizer(Combined, "|");
        tuple = st.nextToken();
        st = new StringTokenizer(tuple, "$");
        temp1 = st.nextToken();
        temp2 = st.nextToken();
        StartSet = union(StartSet, temp1 + "$" + temp2);
        Combined = subtract(Combined);
        spawn = spawn(temp1, States1, Messages1, Transitions1);
    }
}
if (spawn != null) {
    st = new StringTokenizer(spawn, "\$"),
    spawn_messages1 = new String[st.countTokens()];
    while (st.hasMoreTokens()) {
      spawn_messages1[i] = st.nextToken();
      i++;
    }
    i = 0;
  }
  spawn = spawn(temp2, States2, Messages2, Transitions2);
  if (spawn != null) {
    st = new StringTokenizer(spawn, "\$"),
    spawn_messages2 = new String[st.countTokens()];
    while (st.hasMoreTokens()) {
      spawn_messages2[i] = st.nextToken();
      i++;
    }
    i = 0;
  }
  for (i = 0; i < spawn_messages1.length; i++)
    for (j = 0; j < spawn_messages2.length; j++) {
      dir1 = getDirection(temp1, spawn_messages1[i], States1, Messages1, Transitions1);
      dir2 = getDirection(temp2, spawn_messages2[j], States2, Messages2, Transitions2);
      if ((dir1 != null) && (dir2 != null)) {
        if ((dir1.equals("IN") && dir2.equals("IN"))
            || (dir1.equals("OUT") &&
                dir2.equals("OUT"))) {
          element1 = TimeSlice1[index(States1, temp1)][index(Messages1, spawn_messages1[i])];
          element2 = TimeSlice2[index(States2, temp2)][index(Messages2, spawn_messages2[j])];
          st = new StringTokenizer(element1, "\|"),
          array1 = new String[st.countTokens()];
          while (st.hasMoreTokens()) {
            array1[ctr] = st.nextToken();
            ctr++;
          }
        }
      }
    }
Intersection.java:

```java
ctr = 0;
"|");
String[st.countTokens()];

array2.length; jin++) {
getTimize(array1[jin]);
getTimize(array2[jin]);
getState(array1[jin]);
getState(array2[jin]);
slice2[0] <= min(
    slice1[1], slice2[1])) { 
    min(slice1[1], slice2[1]);
    max(slice1[0], slice2[0]);
} != maximum)
```

(Continued) Intersection.java:

```java
ctr = 0;
|");
String[st.countTokens()];

array2 = new String[st.countTokens()];
while (st.hasMoreTokens()) {
    array2[ctr] = st.nextToken();
    ctr++;
}
ctr = 0;
for (iin = 0; iin < array1.length; iin++)
    for (jin = 0; jin < array2.length; jin++) {
        slice1 =
        slice2 =
        state1 =
        state2 =
        if (max(slice1[0], slice2[0]) <= min(slice1[1], slice2[1])) {
            minimum =
            maximum =
            if (minimum != maximum)
```
(Continued) Intersection.java:

```java
InteractionTimeSlice[index(States1, temp1)][index(States2, temp2)][index(Messages1, spawn_messages1[i])] = union(
    InteractionTimeSlice[index(States1, temp1)][index(States2, temp2)][index(Messages1, spawn_messages1[i])],
    "[" + maximum + "," + minimum
    + "]" + state1 + "$" + state2);
    
    //System.out.println(state1+state2);
    Combined = union(Combined, state1 + "$" + state2);
    max
    }
    }
    } // if spawn ends
    } // if dir null check ends
    } // outermost for ends
    i = 0;
    j = 0;
}
while ends
st = new StringTokenizer(StartSet, "|");
while (st.hasMoreTokens()) {
    tuple = st.nextToken();
    st1 = new StringTokenizer(tuple, "$");
    temp1 = st1.nextToken();
    temp2 = st1.nextToken();
```
(Continued) Intersection.java:

```java
if (intersect(StateSet1[index(States1, temp1)], EndStates1)
    && intersect(StateSet2[index(States2, temp2)],
        EndStates2)) {
    for (i = 0; i < InteractionTimeSlice[0][0].length; i++)
        {
            if (InteractionTimeSlice[index(States1, temp1)][index(States2, temp2)][i] != null)
                falseAlarm = true;

            if (falseAlarm == false)
                {
                    FinishSet = union(FinishSet, temp1 + "$" + temp2);
                }
            falseAlarm = false;
        }
} //end of while tokens

public void beginCalculation(String filename1, String filename2) {
    InteractionTrace it = new InteractionTrace();
    it.doInitialSetup(filename1, filename2);
    it.calculateIntersection();
}

public void printInteractionTrace() {
    print32(this.InteractionTimeSlice);
}

public String getStartSet() {
    return StartSet;
}

public String setStartSet(String s) {
    StartSet = s;
}

public String getInitial() {
    return Initial;
}

public String setInitial(String s) {
    Initial = s;
}

public String getFinishSet() {
    return FinishSet;
}

public String setFinishSet(String s) {
    FinishSet = s;
}

public String[][][] getIntersectionTimeSlice() {
    return IntersectionTimeSlice;
}

public void setIntersectionTimeSlice(String[][][] array) {
    InteractionTimeSlice = array;
}

public String[][] getTimeSlice1() {
    return TimeSlice1;
}

public String[][] getTimeSlice2() {
    return TimeSlice2;
}
```
/* Check if two TICs are equivalent (both for compatibility and replaceability) */
* @author: Omkar Jayant Tilak
* @date: Aug 2006
* */

class Equivalent {
    private TemporalInteractionContract tic1;
    private TemporalInteractionContract tic2;
    private static TimeSlice ts1;
    private Initialize init1;
    private Initialize init2;
    private static String[] States1;
    private static String[] StartState1;
    private static String[] States2;
    private static String[] StartState2;
    private static String[] Messages1;
    private static String[] Messages2;
    private static String[][] Transitions1;
    private static String[][] Transitions2;
    private static String[] TimeSlice1;
    private static String[] TimeSlice2;
    private static String Combined;
    private static String StartSet;
    Equivalent() {
        tic1 = null;
        tic2 = null;
        ts1 = null;
        init1 = null;
        init2 = null;
        States1 = null;
        StartState1 = null;
        States2 = null;
        StartState2 = null;
        Messages1 = null;
        Messages2 = null;
        Transitions1 = null;
        Transitions2 = null;
        TimeSlice1 = null;
        TimeSlice2 = null;
        Combined = null;
        StartSet = null;
    }
}

package code;

import java.util.StringTokenizer;

public class Equivalent {
    private TemporalInteractionContract tic1;
    private TemporalInteractionContract tic2;
    private static TimeSlice ts1;
    private Initialize init1;
    private Initialize init2;
    private static String[] States1;
    private static String[] StartState1;
    private static String[] States2;
    private static String[] StartState2;
    private static String[] Messages1;
    private static String[] Messages2;
    private static String[][] Transitions1;
    private static String[][] Transitions2;
    private static String[] TimeSlice1;
    private static String[] TimeSlice2;
    private static String Combined;
    private static String StartSet;
    Equivalent() {
        tic1 = null;
        tic2 = null;
        ts1 = null;
        init1 = null;
        init2 = null;
        States1 = null;
        StartState1 = null;
        States2 = null;
        StartState2 = null;
        Messages1 = null;
        Messages2 = null;
        Transitions1 = null;
        Transitions2 = null;
        TimeSlice1 = null;
        TimeSlice2 = null;
        Combined = null;
        StartSet = null;
    }
}
(Continued) Equivalent.java:

```java
private static String getState(String s) {
    String temp = null, result = null;
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "$");
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (!(temp.startsWith("[")) {
                result = temp;
            }
        }
    }
    return result;
}

private static String getStateSpecial(String s) {
    String temp = null, result = null;
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "$");
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (!(temp.startsWith("[")))
                if (result == null)
                    result = temp;
                else
                    result = result + "$" + temp;
        }
    }
    return result;
}

static float[][] sort2D(float a[][]) {
    float tmp;
    int columnSize = a[0].length;
    int i, j;
    for (i = 0; i < columnSize - 1; i++) {
        for (j = 0; j < columnSize - 1 - i; j++)
            if (a[0][j + 1] < a[0][j]) { /* compare the two neighbors */
                tmp = a[0][j]; /* swap a[j] and a[j+1] */
                a[0][j] = a[0][j + 1];
                a[0][j + 1] = tmp;
                tmp = a[1][j]; /* swap a[j] and a[j+1] */
                a[1][j] = a[1][j + 1];
                a[1][j + 1] = tmp;
            }
    }
    return a;
}
```
(Continued) Equivalent.java:

```java
private static float toFloat(String s) {
    float f = (float) 0;
    if (s == null)
        return f;
    if (s.equals("INFINITY"))
        f = Float.POSITIVE_INFINITY;
    else if (s != null)
        f = Float.parseFloat(s);
    return f;
}

private static int index(String yourArray[], String tobefound) {
    final int notfound = -1;
    int counter;
    for (counter = 0; counter < yourArray.length; counter++) {
        if (tobefound.equals(yourArray[counter])) {
            return counter;
        }
    } // end for loop
    return notfound;
}

private static float min(float a, float b) {
    if (a < b)
        return a;
    else
        return b;
}

private static float max(float a, float b) {
    if (a > b)
        return a;
    else
        return b;
}
```
static boolean similar(String a, String b) {
    boolean result = false;
    String[] astring = null;
    String[] bstring = null;
    int i = 0;

    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, "\$" );
        astring = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            astring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;

    if (b != null) {
        StringTokenizer st = new StringTokenizer(b, "\$" );
        bstring = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            bstring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;

    if ((a != null) && (b != null)) {
        if (astring[0].equals(bstring[0]))
            return true;
    }

    if ((a == null) && (b == null))
        result = false;
    return result;
}

String spawnTimeSlice(String[][] TimeSlice, String[] States, String[] Messages, String state) {
    String result = null;
    for (int i = 0; i < Messages.length; i++) {
        if (TimeSlice[index(States, state)][i] != null) {
            if (result == null)
                result = Messages[i];
            else
                result = result + "$" + Messages[i];
        }
    return result;
}
(Continued) Equivalent.java:

```java
private static String union(String a, String b) {
    String result = null;
    if ((a == null) && (b != null))
        result = b;
    else if ((a != null) && (b == null))
        result = a;
    else if ((a != null) && (b != null))
        result = a + "|" + b;
    return result;
}

private static String getDirection(String state, String message,
                                   String[] States, String[] Messages, String[][][] Transitions) {
    String direction = null;
    for (int i = 0; i < States.length; i++) {
        if (Transitions[index(States, state)][i][index(Messages, message)] != null) {
            StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][i][index(Messages, message)], 
"$");
            while (st.hasMoreTokens()) {
                direction = st.nextToken();
                st.nextToken();
            }
        }
    }
    return direction;
}

static String subtract(String a) {
    String result = null;
    String[] astring = null;
    int i = 0;
    if (a != null) {
        StringTokenizer st = new StringTokenizer(a, 
"|");
        astring = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            astring[i] = st.nextToken();
            i++;
        }
    }
    i = 0;
```
(Continued) Equivalent.java:

```java
for (i = 1; i < astring.length; i++) {
    if (result == null)
        result = astring[i];
    else
        result = result + "|" + astring[i];
}
return result;
}

private static float[] stringFloat(String s) {
    String temp = null, part1, part2;
    int index;
    float[] result = new float[2];
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, "$");  
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            if (temp.startsWith("[")) {
                index = temp.indexOf(",");
                part1 = temp.substring(1, index);
                part2 = temp.substring(index + 1, temp.length() - 1);
                result[0] = toFloat(part1);
                result[1] = toFloat(part2);
            }
        }
    }
    return result;
}

static float[][] toArray(String s) {
    float[] temp = new float[2];
    String temp1;
    float[][] result = null;
    int i = 0;
    if (s != null) {
        StringTokenizer st = new StringTokenizer(s, ",");
        result = new float[2][st.countTokens()];
        while (st.hasMoreTokens()) {
            temp1 = st.nextToken();
            temp = stringFloat(temp1);
            result[0][i] = temp[0];
            result[1][i] = temp[1];
            i++;
        }
    }
    return result;
}
```
(Continued) Equivalent.java:

```java
private static String spawn(String state, String[] States,
String[] Messages, String[][][] Transitions) {
    String direction = null;
    String duration = null;
    String result = null;

    for (int j = 0; j < States.length; j++)
        for (int k = 0; k < Messages.length; k++) {
            if ((Transitions[index(States, state)][j][k] != null)) {
                StringTokenizer st = new StringTokenizer(Transitions[index(States, state)][j][k], "$");
                while (st.hasMoreTokens()) {
                    direction = st.nextToken();
                    duration = st.nextToken();
                }
                if (duration.equals("NO")) {
                    if (result == null)
                        result = Messages[k];
                    else
                        result = result + "$" + Messages[k];
                }
            }
        }
    return result;
}

static boolean contains(String TimeSlice, String element) {
    float[][] temp1;
    float[] temp2 = new float[2];
    float[] temp3 = new float[2];
    float min, max;
    StringTokenizer st;
    boolean result = false, flag = true;
    int size = 0, i = 0, istore = -1;
    String tokens;
    st = new StringTokenizer(TimeSlice, "|");
    size = st.countTokens();
    temp1 = new float[2][size];
    temp1 = sort2D(temp1);
    temp2 = stringFloat(element);
    int columnSize = temp1[0].length;
```
st = new StringTokenizer(TimeSlice, "|");
while (st.hasMoreTokens()) {
    tokens = st.nextToken();
    if (similar(tokens, element))
        // if(tokens.equals(element))
        return true;
    temp3 = stringFloat(tokens);
    if (temp3[0] <= temp2[0] && temp3[1] >= temp2[1])
        return true;
}
if (columnSize > 1) {
    min = temp1[0][0];
    max = temp1[1][size - 1];
    if ((temp2[0] < min) || (temp2[1] > max)) {
        return result;
    }
    for (i = 0; i < size; i++) {
        if ((temp1[0][i] <= temp2[0]) && (temp1[1][i] <= temp2[1]))
            istore = i;
    }
    i = istore;
    if (i == -1) {
        return result;
    }
    while (flag) {
        if ((i + 1) < (size)) {
            if ((temp1[0][i] <= temp1[0][i + 1])
                && (temp1[1][i] >= temp1[0][i + 1])) {
                if (temp1[1][i + 1] >= temp2[1]) {
                    result = true;
                    flag = false;
                    i++;
                } else
                    flag = false;
            } else
                flag = false;
        }
    }
}
else if (columnSize == 1) {
    min = temp1[0][0];
    max = temp1[1][0];
    if ((min <= temp2[0]) && (max >= temp2[1]))
        result = true;
}
return result;
}

void initializeVariables(String filename1, String filename2) {
    String[] temp = null;
    int i, j, maxstates, maxmessages;

    tic1 = new TemporalInteractionContract();
    tic2 = new TemporalInteractionContract();

    ts1 = new TimeSlice();

    init1 = new Initialize();
    init1.doInitialize(filename1);
    tic1 = init1.getTemporalInteractionContract();
    States1 = tic1.getStates();
    StartState1 = tic1.getStartState();
    Messages1 = tic1.getMessages();
    Transitions1 = tic1.getTransitions();
    TimeSlice1 = new String[States1.length][Messages1.length];
    ts1.startProcessing(filename1);
    temp = new String[Messages1.length];
    for (i = 0; i < States1.length; i++) {
        temp = ts1.getTimeSlice(States1[i]);
        for (j = 0; j < temp.length; j++)
            TimeSlice1[i][j] = temp[j];
    }

    init2 = new Initialize();
    init2.doInitialize(filename2);
    tic2 = init2.getTemporalInteractionContract();
    States2 = tic2.getStates();
    StartState2 = tic2.getStartState();
    Messages2 = tic2.getMessages();
    Transitions2 = tic2.getTransitions();
    TimeSlice2 = new String[States2.length][Messages2.length];
    ts1.startProcessing(filename2);
    temp = new String[Messages2.length];
    for (i = 0; i < States2.length; i++) {
        temp = ts1.getTimeSlice(States2[i]);
        for (j = 0; j < temp.length; j++)
            TimeSlice2[i][j] = temp[j];
    }
boolean checkEquivalence(String checkType) {
    String temp1, temp2, tuple, spawn, state1, state2, dir1, dir2, nextToken1, 
    nextToken2, temp;
    String[] spawn_messages1 = null;
    String[] spawn_messages2 = null;
    String element1 = null, element2 = null;
    StringTokenizer st, st1;
    boolean equivalent = true;
    boolean enter = false;
    boolean same_message = false;
    int i = 0, j = 0, ctr = 0, iin = 0, jin = 0;
    float[] slice1 = new float[2];
    float[] slice2 = new float[2];
    Combined = StartState1[0] + "$" + StartState2[0];
    while (Combined != null) {
        st = new StringTokenizer(Combined, "|");
        tuple = st.nextToken();
        st = new StringTokenizer(tuple, "$");
        temp1 = st.nextToken();
        temp2 = st.nextToken();
        if (index(States1, temp1) == -1) {
            temp = temp1;
            temp1 = temp2;
            temp2 = temp;
        }
        StartSet = union(StartSet, temp1 + "$" + temp2);
        Combined = subtract(Combined);
        spawn = spawnTimeSlice(TimeSlice1, States1, Messages1, temp1);
        if (spawn != null) {
            st = new StringTokenizer(spawn, ";");
            spawn_messages1 = new String[st.countTokens()];
            while (st.hasMoreTokens()) {
                spawn_messages1[i] = st.nextToken();
                i++;
            }
            i = 0;
        }
        System.out.println("Analysis for state "+temp1 
    + ";" from first TIC and "+temp2 
    + ";" from second TIC");
        spawn = spawnTimeSlice(TimeSlice2, 
        States2, Messages2, temp2);
    }
(Continued) Equivalent.java:

```java
if (spawn != null) {
    st = new StringTokenizer(spawn, "$");
    spawn_messages2 = new String[st.countTokens()];
    while (st.hasMoreTokens()) {
        spawn_messages2[i] = st.nextToken();
        i++;
    }
    i = 0;
    for (i = 0; i < spawn_messages1.length; i++)
        for (j = 0; j < spawn_messages2.length; j++) {
            dir1 = getDirection(temp1, spawn_messages1[i],
                             Messages1, States1, Transitions1);
            dir2 = getDirection(temp2, spawn_messages2[j],
                             Messages2, States2, Transitions2);
            same_message = spawn_messages1[i].equals(spawn_messages2[j]);
            if ((dir1 != null) && (dir2 != null) && same_message) {
                System.out.println("For message "+ spawn_messages1[i] + "-");
                if (checkType.equals("compatibility") ||
                    checkType.equals("Compatibility"))
                    enter = (dir1.equals("IN") &&
                             (dir1.equals("OUT") && dir2.equals("IN")));  |
                if (checkType.equals("replaceability") ||
                    checkType.equals("Replaceability"))
                    enter = (dir1.equals("IN") &&
                             (dir1.equals("OUT") && dir2.equals("OUT")));  |
                if (checkType.equals("compatibility") ||
                    checkType.equals("Compatibility"))
                    enter = (dir1.equals("IN") &&
                             (dir1.equals("OUT") && dir2.equals("OUT")));  |
            }
        }
    }
```

if (enter) {
    if (checkType.equals("compatibility") ||
        checkType.equals("Compatibility")) {
        if (dir1.equals("IN") && dir2.equals("OUT")) {
            element1 =
                TimeSlice2[index(States2, temp2)][index(
                    Messages2, spawn_messages2[j]]));
            element2 =
                TimeSlice1[index(States1, temp1)][index(
                    Messages1, spawn_messages1[i]]));
            if (dir1.equals("OUT") &&
                dir2.equals("IN")) {
                element1 =
                    TimeSlice1[index(States1, temp1)][index(
                        Messages1, spawn_messages1[i]]));
                element2 =
                    TimeSlice2[index(States2, temp2)][index(
                        Messages2, spawn_messages2[j]]));
                } else if (dir1.equals("OUT")
                    &&
                    dir2.equals("IN")) {
                element1 =
                    TimeSlice1[index(States1, temp1)][index(
                        Messages1, spawn_messages1[i]]));
                element2 =
                    TimeSlice2[index(States2, temp2)][index(
                        Messages2, spawn_messages2[j]]));
                } else if (checkType.equals("replaceability") ||
                    checkType.equals("Replaceability")) {
                element1 =
                    TimeSlice1[index(States1, temp1)][index(
                        Messages1, spawn_messages1[i]]));
                element2 =
                    TimeSlice2[index(States2, temp2)][index(
                        Messages2, spawn_messages2[j]]));
            }
        }
    }
}
if (element1 != null) {
    st = new StringTokenizer(element1, "|");
    st.nextToken();

    while (st.hasMoreTokens()) {
        nextToken1 = st.nextToken();

        if (contains(element2, nextToken1)) {
            System.out.println(element2 + " contains " + nextToken1);
            equivalent = true; do nothing
        } else {
            System.out.println("------------------------------------------------------------");
            System.out.println("ERROR!!!!! " + element2 + " does not contain " + nextToken1);
            System.out.println("------------------------------------------------------------");
            equivalent = false;
        }
    }
}
(Continued) Equivalent.java:

```java
st1 = new StringTokenizer(element2, "|");
while (st1.hasMoreTokens()) {
    nextToken2 = st1.nextToken();
    slice2 = stringFloat(nextToken2);
    slice1 = stringFloat(nextToken1);
    if (max(slice1[0], slice2[0]) <= min(
        slice1[1], slice2[1])) {
        Combined = union(
            Combined,
            getState(nextToken1)
            + "$"
            + getState(nextToken2));
    }
    } // while st1 ends
    } // while st ends
} // end dir if
} // end outer if
} // end for
i = 0;
System.out.println();
} // end while
return equivalent;
}

public boolean isEquivalent(String filename1, String filename2,
String checkType) {
    boolean result = false;
    Equivalent e = new Equivalent();
e.initializeVariables(filename1, filename2);
    result = e.checkEquivalence(checkType);
    return result;
}
```
CheckCompatibility.java:

```java
package code;

import java.util.StringTokenizer;

public class CheckCompatibility {

    static String[][] InteractionTimeSlice = null;
    static String[] StartState;
    static String FinishSet;
    static String[][] TimeSlice1 = null;
    static String[] States1 = null;
    static String[] States2 = null;
    static String[] Messages1 = null;
    static String[] Messages2 = null;
    static String[] MaxStates = null;
    static String[][] MaxMessages = null;
    static String[][] Spawn = null;
    static TemporalInteractionContract tic1 = null;
    static TemporalInteractionContract tic2 = null;
    static Equivalent equiv = null;
    static String checkType = "Compatibility";
    static InteractionTrace inter = new InteractionTrace();
    static void print31(String[][][] array) {
        int x = array.length;
        int y = array[0].length;
        int z = array[0][0].length;
        // System.out.println(x+" "+y+" "+z);
        for (int i = 0; i < x; i++) {
            System.out.println("----------------");
            for (int j = 0; j < y; j++) {
                for (int k = 0; k < z; k++) {
                    System.out.print( " " + array[i][j][k]);
                }
                System.out.println(" \n");
            }
        }
    }
}
```
(Continued) CheckCompatibility.java:

```java
static String[] extractStates(String s, int choice) {
    String[] result = null;
    String temp = null;
    StringTokenizer st;
    StringTokenizer st1;
    int i = 0;

    if (s != null) {
        st = new StringTokenizer(s, "|");
        result = new String[st.countTokens()];
        while (st.hasMoreTokens()) {
            temp = st.nextToken();
            result[i] = temp;
            i++;
        }
    }
    return result;
}

static String[][] getSpawn(String[][][] TimeSlice, String[] MaxStates,
                         String[] MaxMessages) {
    String[][] result = new String[MaxStates.length][MaxStates.length];
    StringTokenizer st;
    StringTokenizer st1;
    String temp;
    String temp1;
    for (int i = 0; i < MaxStates.length; i++)
        for (int j = 0; j < MaxStates.length; j++)
            for (int k = 0; k < MaxMessages.length; k++) {
                if (TimeSlice[i][j][k] != null) {
                    st = new StringTokenizer(TimeSlice[i][j][k], "|");
                    while (st.hasMoreTokens()) {
                        temp = st.nextToken();
                        st1 = new StringTokenizer(temp, "$");
                        temp1 = st1.nextToken();
                        if ((temp1.charAt(1) == '0')
                            && (temp1.charAt(2) == '.
                            && (temp1.charAt(3) == '0')) {
                            if (result[i][j] == null)
                                result[i][j] = MaxMessages[k];
                        }
                    }
                }
            }
    return result;
```
(Continued) CheckCompatibility.java:

```java
else
    result[i][j] = result[i][j] + "$" + MaxMessages[k];
}
st1.nextToken();
st1.nextToken();
}
// if ends
// for ends
return result;
}
static boolean determineCompatibility(String filename1, String filename2, String type) {
    boolean result = false;
    int x, y, z;
    if (type.equals("Partial") || type.equals("partial")) {
        inter.beginCalculation(filename1, filename2);
        InteractionTimeSlice = inter.getIntersectionTimeSlice();
        if (InteractionTimeSlice != null) {
            x = InteractionTimeSlice.length;
            y = InteractionTimeSlice[0].length;
            z = InteractionTimeSlice[0][0].length;
            for (int i = 0; i < x; i++)
                for (int j = 0; j < y; j++)
                    for (int k = 0; k < z; k++) {
                        if (InteractionTimeSlice[i][j][k] != null) {
                            result = true;
                            return result;
                        }
                    }
        }
    } // if partial end
    if (type.equals("Complete") || type.equals("complete")) {
        equiv = new Equivalent();
        result = equiv.isEquivalent(filename1, filename2, checkType);
    } // if complete ends
    return result;
```
(Continued) CheckCompatibility.java:

```java
public static void main(String[] args) {
    boolean result = false;
    System.out.println("PARTIAL COMPATIBILITY ANALYSIS");
    System.out.println("="");
    System.out.println();
    result = determineCompatibility("C:/Documents and Settings/Einstein/Desktop/UniFrame/" + args[0] + ".xml",
                                   "C:/Documents and Settings/Einstein/Desktop/UniFrame/
                                   + args[1] + ".xml", "partial");

    if (result == true) {
        System.out.println("Since InteractionTrace of these two TICs is not null, these two TICs ARE PARTIALLY COMPATIBLE");
        System.out.println("The non-null elements of the InteractionTrace are as follows:");
        inter.printInteractionTrace();
    }
    if (result == false) {
        System.out.println("Since InteractionTrace of these two TICs is null, these two TICs ARE NOT PARTIALLY COMPATIBLE");
    }

    System.out.println();
    System.out.println("COMPLETE COMPATIBILITY ANALYSIS");
    System.out.println("="");
    System.out.println();
    result = determineCompatibility("C:/Documents and Settings/Einstein/Desktop/UniFrame/" + args[0] + ".xml",
                                   "C:/Documents and Settings/Einstein/Desktop/UniFrame/
                                   + args[1] + ".xml", "complete");

    if (result == true) {
        System.out.println("Since no errors were discovered during analysis, these two TICs ARE COMPLETELY COMPATIBLE");
        System.out.println("="");
    }
}
```

(Continued) CheckCompatibility.java:

```java
if (result == false) {
    System.out.println("Since errors were discovered during analysis, these two TICs ARE NOT COMPLETELY COMPATIBLE");
    System.out.println("Please fix the indicated errors to make these two TICs completely compatible with each other.");
    System.out.println("==================================================");
    }
}
```
package code;

import java.util.StringTokenizer;

public class CheckReplaceability {
    static String[][] IntersectionTimeSlice = null;
    static String[] StartState;
    static String FinishSet;
    static String[][] TimeSlice1 = null;
    static String[] States1 = null;
    static String[] States2 = null;
    static String[] Messages1 = null;
    static String[] Messages2 = null;
    static String[] MaxStates = null;
    static String[] MaxMessages = null;
    static String[][] Spawn = null;
    static TemporalInteractionContract tic1 = null;
    static TemporalInteractionContract tic2 = null;
    static Equivalent equiv = null;
    static String checkType = "Replaceability";
    static Intersection inter = new Intersection();
    static String[] extractStates(String s, int choice) {
        String[] result = null;
        String temp = null;
        StringTokenizer st;
        StringTokenizer st1;
        int i = 0;

        if (s != null) {
            st = new StringTokenizer(s, "\|");
            result = new String[st.countTokens()];
            while (st.hasMoreTokens()) {
                temp = st.nextToken();
                result[i] = temp;
                i++;
            }
        }
        return result;
    }
}
(Continued) CheckReplaceability.java:

```java
static String[][] getSpawn(String[][][] TimeSlice, String[] MaxStates, String[] MaxMessages) {
    String[][] result = new String[MaxStates.length][MaxStates.length];
    StringTokenizer st;
    StringTokenizer st1;
    String temp;
    String temp1;
    for (int i = 0; i < MaxStates.length; i++)
        for (int j = 0; j < MaxStates.length; j++)
            for (int k = 0; k < MaxMessages.length; k++) {
                if (TimeSlice[i][j][k] != null) {
                    st = new StringTokenizer(TimeSlice[i][j][k], "|");
                    while (st.hasMoreTokens()) {
                        temp = st.nextToken();
                        st1 = new StringTokenizer(temp, "$");
                        temp1 = st1.nextToken();
                        if ((temp1.charAt(1) == '0')
                            && (temp1.charAt(2) == '.')
                            && (temp1.charAt(3) == '0')) {
                            if (result[i][j] == null)
                                result[i][j] = MaxMessages[k];
                            else
                                result[i][j] = result[i][j] + "$" + MaxMessages[k];
                        }
                        st1.nextToken();
                    }
                }
            }
    return result;
}
```
(Continued) CheckReplaceability.java:

```java
static boolean determineReplaceability(String filename1, String filename2,
                                        String type) {
    boolean result = false;

    int x, y, z;

    if (type.equals("Partial") || type.equals("partial")) {
        inter.beginCalculation(filename1, filename2);
        IntersectionTimeSlice = inter.getIntersectionTimeSlice();

        if (IntersectionTimeSlice != null) {
            x = IntersectionTimeSlice.length;
            y = IntersectionTimeSlice[0].length;
            z = IntersectionTimeSlice[0][0].length;
            // print3(IntersectionTimeSlice);
            for (int i = 0; i < x; i++)
                for (int j = 0; j < y; j++)
                    for (int k = 0; k < z; k++) {
                        if (IntersectionTimeSlice[i][j][k] !=
                            null) {
                            result = true;
                            return result;
                        }
                    }
    }

    if (type.equals("Complete") || type.equals("complete")) {
        String[][] ModifiedTimeSlice;
        equiv = new Equivalent();
        result = equiv.isEquivalent(filename1, filename2, checkType);
    }

    return result;
}

public static void main(String[] args) {
    boolean result = false;
    System.out.println("PARTIAL REPLACEABILITY ANALYSIS");
    System.out.println("================================");
    System.out.println();
    result = determineReplaceability(
        "C:/Documents and Settings/Einstein/Desktop/UniFrame/" + args[0] + ".xml",
```
(Continued) CheckReplaceability.java:

```java
if (result == true) {
    System.out.println("Since IntersectionTrace of these two TICs is not null, these two TICs ARE PARTIALLY REPLACEABLE");
    System.out.println("The non-null elements of the IntersectionTrace are as follows:");
    inter.printIntersectionTrace();
} else {
    System.out.println("Since IntersectionTrace of these two TICs is null, these two TICs ARE NOT PARTIALLY REPLACEABLE");
}
System.out.println("COMPLETE REPLACEABILITY ANALYSIS");
System.out.println("=================================");
result = determineReplaceability(
    "C:/Documents and Settings/Einstein/Desktop/UniFrame/" + args[0] + ".xml",
if (result == true) {
    System.out.println("Since no errors were discovered during analysis, these two TICs ARE COMPLETELY REPLACEABLE");
} else {
    System.out.println("Since errors were discovered during analysis, these two TICs ARE NOT COMPLETELY REPLACEABLE");
    System.out.println("Please fix the indicated errors to make these two TICs completely replaceable with each other.");
}
}
```
Server_simulator.java:

```java
package code;

import java.io.*;
import java.net.*;
import java.util.*;

class Server_simulator
{
    static Socket connection;
    static BufferedReader inClient;
    static String current_state;
    static String current_message;
    static String temporal_message;
    static String next_state;
    static int duration;
    static Timer timer = null;
    static String sentence;
    static TemporalInteractionContract tic;
    static String[] States;
    static String[] Start;
    static String[] Messages;
    static String[][][] Transitions;
    static String[] EndStates;
    private static int index(String yourArray[], String tobefound)
    {
        final int notfound = -1;
        int counter;
        for (counter = 0; counter < yourArray.length; counter++) {
            if (tobefound.equals(yourArray[counter]))
            {
                return counter;
            }
        } // end for loop
        return notfound;
    }
}
```
(Continued) Server_simulator.java:

```java
static boolean hasTemporal(String state)
{
    boolean result = false;
    int i,j;
    String temp;

    for(i=0;i<States.length;i++)
        for(j=0;j<Messages.length;j++)
        {
            if(Transitions[index(States,state)][i][j]!=null)
                {
                    StringTokenizer st = new StringTokenizer(Transitions[index(States,state)][i][j], "$");
                    while (st.hasMoreTokens()) {
                        temp = st.nextToken();
                        if(!(temp.equals("NO"))) result = true;
                    }
                }
    return result;
}

static String temporalMessage(String state)
{
    String result = null;
    int i,j;
    String temp;

    for(i=0;i<States.length;i++)
        for(j=0;j<Messages.length;j++)
        {
            if(Transitions[index(States,state)][i][j]!=null)
                {
                    StringTokenizer st = new StringTokenizer(Transitions[index(States,state)][i][j], "$");
                    while (st.hasMoreTokens()) {
                        temp = st.nextToken();
                        if(!(temp.equals("NO"))) result = Messages[j];
                    }
                }
    return result;
}
```
(Continued) Server_simulator.java:

```java
static int temporalValue(String state)
{
    String result1 = null;
    int result = 0;
    Integer result2;
    int i,j;
    String temp;
    for(i=0;i<States.length;i++)
        for(j=0;j<Messages.length;j++)
            {
                if(Transitions[index(States,state)][i][j]!=null)
                {
                    StringTokenizer st = new StringTokenizer(Transitions[index(States,state)][i][j], "$");
                    while (st.hasMoreTokens()) {
                        temp = st.nextToken();
                        if(!(temp.equals("NO"))){
                            result1 = temp;
                            result2 = new Integer(result1);
                            result = result2.intValue();
                        }
                    }
                }
            }
    return result;
}

static String targetState(String state, String message)
{
    int i;
    String result = null;
    for(i=0;i<States.length;i++)
        if(Transitions[index(States,state)][i][index(Messages,message)] != null)
            {
                result = States[i];
            }
    if (result == null)
        {
            result = "ERROR!!! The state \" + state + "\" can not receive the message \" + message + "\";"
        }
    return result;
}
```
(Continued) Server_simulator.java:

```java
public static void main(String args[]) throws Exception
{
    boolean end = false;
    class RemindTask extends TimerTask {
        String local_state;
        boolean end = false;
        RemindTask(String state)
        {
            local_state = state;
        }
        public void run()
        {
            current_state = local_state;
            System.out.println("current state is "+ current_state);
            if(index(EndStates,current_state) != -1) {System.out.println("End state is reached");}
            while(!end)
            if(hasTemporal(current_state))
            {
                temporal_message = temporalMessage(current_state);
                next_state = targetState(current_state,temporal_message);
                duration = temporalValue(current_state);
                timer = new Timer();
                timer.schedule(new RemindTask(next_state), duration*1000);
                end = true;
                try
                {
                    while ((sentence = inClient.readLine()) == null) {
                        sentence = inClient.readLine();
                    }
                    timer.cancel();
                    end = false;
                    current_message = sentence;
                    System.out.println("current message received is " +
                    current_message);
                    current_state =
targetState(current_state,current_message);
                    System.out.println("current state is "+ current_state);
                    if(index(EndStates,current_state) != -1)
                    {System.out.println("End state is reached");}
                    sentence = null;
                }
            
                catch(Exception e)
                {
                    //end if hasTemporal
                }
```

```
if(!hasTemporal(current_state))
{
    try
    {
        while ((sentence = inClient.readLine()) == null) {
            sentence = inClient.readLine();
        }
        current_message = sentence;
        System.out.println("current message received is "+ current_message);
        current_state =
targetState(current_state, current_message);
        end = false;
        System.out.println("current state is "+ current_state);
        if(index(EndStates, current_state) != -1)
        {System.out.println("End state is reached");}
        sentence = null;
    } catch(Exception e){}
    } //end while
} //end while !end
}

tic = new TemporalInteractionContract();
tic = tic.startParsing("C:/Documents and Settings/Einstein/Desktop/UniFrame/ + args[0] + ".xml");
System.out.println("Starting the simulation for" + args[0] + " component");

States = tic.getStates();
Start = tic.getStartState();
Messages = tic.getMessages();
Transitions = tic.getTransitions();
EndStates = tic.getEndStates();

current_state = Start[0];

// Create the timer and schedule the task for n seconds ahead.
ServerSocket welcome = new ServerSocket(2000);
Server_simulator.java:

```java
while(true)
{
    connection = welcome.accept();
inClient = new BufferedReader(new InputStreamReader(connection.getInputStream()));

    DataOutputStream out = new DataOutputStream(connection.getOutputStream());
    System.out.println("current state is "+current_state);
    if(index(EndStates,current_state) != -1) {System.out.println("End state is reached");System.exit(0);} 
    while(!end)
    {
        if(hasTemporal(current_state))
        {
            temporal_message = temporalMessage(current_state);
            next_state = targetState(current_state,temporal_message);
            duration = temporalValue(current_state);
            System.out.println("Duration is "+duration);
            timer = new Timer();
timer.schedule(new RemindTask(next_state), duration*1000);
            end = true;
            while (((sentence = inClient.readLine()) == null)) {
                sentence = inClient.readLine();
            }
            timer.cancel();
            end = false;
            current_message = sentence;
            System.out.println("current message is "+current_message);
            current_state = targetState(current_state,current_message);
            System.out.println("current state is "+current_state);
        }
    } //end if hasTemporal
```
(Continued) Server_simulator.java:

```java
if(!hasTemporal(current_state))
{
    end = false;
    while ((sentence = inClient.readLine()) == null) {
        sentence = inClient.readLine();
    }
    current_message = sentence;
    System.out.println("current message received is " +
current_message);
    current_state =
targetState(current_state,current_message);
    System.out.println("current state is " + current_state);
    if(index(EndStates,current_state) != -1)
    {
        System.out.println("End state is reached");
        sentence = null;
        //end while hasTemporal
    }
    //end while !end
}
//end while true
//end main
```

Client_simulator.java:

```java
/*
 * Simulates the client behavior.
 * @author: Omkar Jayant Tilak
 * @date: Aug 2006
 * */

package code;

import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

class Client_simulator
{
    static Socket client;
    static DataOutputStream out;
    static String current_state;
    static String current_message;
    static String temporary_message;
    static String next_state;
    static int duration;
    static Timer timer = null;
    static String sentence;
    static TemporalInteractionContract tic;
    static String[] States;
    static String[] Start;
    static String[] Messages;
    static String[][] Transitions;
    static String[] EndStates;
    static Random r = new Random();
    static int randomfac = 8;
    static boolean use_rand = false;

    private static int index(String yourArray[], String tobefound) {
        final int notfound = -1;
        int counter;
        for (counter = 0; counter < yourArray.length; counter++) {
            if (tobefound.equals(yourArray[counter])) {
                return counter;
            }
        } // end for loop
        return notfound;
    }
}
```
static boolean hasTemporal(String state) {
    boolean result = false;
    int i,j;
    String temp;
    for(i=0;i<States.length;i++)
    for(j=0;j<Messages.length;j++)
    {
        if(Transitions[index(States,state)][i][j]!=null)
        {
            StringTokenizer st = new StringTokenizer(Transitions[index(States,state)][i][j], "$");
            while (st.hasMoreTokens()) {
                temp = st.nextToken();
                temp = st.nextToken();
                if(!(temp.equals("NO"))) result = true;
            }
        }
    }
    return result;
}
static String temporalMessage(String state) {
    String result = null;
    int i,j;
    String temp;
    for(i=0;i<States.length;i++)
    for(j=0;j<Messages.length;j++)
    {
        if(Transitions[index(States,state)][i][j]!=null)
        {
            StringTokenizer st = new StringTokenizer(Transitions[index(States,state)][i][j], "$");
            while (st.hasMoreTokens()) {
                temp = st.nextToken();
                temp = st.nextToken();
                if(!((temp.equals("NO"))) result = Messages[j];
            }
        }
    }
    return result;
}
(Continued) Client_simulator.java:

```java
static int temporalValue(String state) {
    String result1 = null;
    int result = 0;
    Integer result2;
    int i,j;
    String temp;
    for(i=0;i<States.length;i++)
        for(j=0;j<Messages.length;j++)
            if(Transitions[index(States,state)][i][j]!=null) {
                StringTokenizer st = new StringTokenizer(Transitions[index(States,state)][i][j], "$");
                while (st.hasMoreTokens()) {
                    temp = st.nextToken();
                    temp = st.nextToken();
                    if(!(temp.equals("NO"))) {
                        result1 = temp;
                        result2 = new Integer(result1);
                        result = result2.intValue();
                    }
                }  
                result1 = temp;
                result2 = new Integer(result1);
                result = result2.intValue();
            }
    return result;
}

static String targetState(String state, String message) {
    int i;
    String result = null;
    for(i=0;i<States.length;i++)
        if(Transitions[index(States,state)][i][index(Messages,message)] != null) {
            result = States[i];
        }
    return result;
}
```
static String transitionMessages(String state) {
    int i,j,k=0;
    String result = null;
    for(i=0;i<States.length;i++)
        for(j=0;j<Messages.length;j++)
            {
                if(Transitions[index(States,state)][i][j] != null &&
                   (!(Messages[j].equals(temporalMessage(state))))
                    {
                    if(k==0) result = Messages[j];
                    else result = result + "$" + Messages[j];
                    k++;
                    }
            }
    return result;
}

public static void main(String args[]) throws Exception {
    boolean end = false;
    String messages;
    String message = null;
    int messagesno;
    int random;
    int i;

    class RemindTask extends TimerTask {
        String local_state;
        boolean end = false;
        String messages;
        String message = null;
        String local_message = null;
        boolean local_flag = false;
        int messagesno;
        int random;
        int i;

        RemindTask(String state) {
            local_state = state;
        }
        RemindTask(String state, String message, boolean flag) {
            local_state = state;
            local_message = message;
            local_flag = flag;
        }
public void run() {
    if(local_flag == false) {
        current_state = local_state;
        System.out.println("current state is " + current_state);
    }
    if(local_flag == true) {
        try {
            out.writeBytes(local_message + 
                System.out.println("current state is " + current_state);
        }
        catch(Exception e) {} } 
        current_state = targetState(local_state, local_message);
        System.out.println("current state is " + current_state);
        } 
        try {client.close();System.out.println("End state is reached");System.exit(0);catch(Exception e){}} 

while(!end) {
    if(hasTemporal(current_state)) {
        if(use_rand) random = r.nextInt(randomfac);
        else random = 1;
        duration = temporalValue(current_state);
        if(random == 0) {
            temporal_message = temporalMessage(current_state);
            next_state = targetState(current_state,temporal_message);
            System.out.println("Duration is" + duration);
            timer = new Timer();
            timer.schedule(new RemindTask(next_state), duration*1000);
            end = true;
        }
        else {
            messages = transitionMessages(current_state);
            StringTokenizer st = new StringTokenizer(messages, "$");
            messagesno = st.countTokens();
            if(messagesno == 0) System.exit(0);
(Continued) Client_simulator.java:

```java
if(messagesno == 1)
    {
        message = st.nextToken();
        System.out.println("current message sent is "+
message);
        timer = new Timer();
        timer.schedule(new RemindTask(current_state,
message, true), (duration
- 1)*1000);
        end = true;
    }
if(messagesno > 1)
    {
        random = r.nextInt(messagesno) + 1;
        for(i=0;i<random;i++)
        {
            message = st.nextToken();
        }
        System.out.println("current message sent is "+
message);
        timer = new Timer();
        timer.schedule(new RemindTask(current_state,
message, true), (duration
- 1)*1000);
        end = true;
    }
//end if random > 0
}
} //end if hasTemporal
if(!hasTemporal(current_state))
    {
        end = false;
        messages = transitionMessages(current_state);
        Stringtokenizer st = new Stringtokenizer(messages,
"\$");
        messagesno = st.countTokens();
        if(messagesno == 0)
            {
                try {client.close();System.exit(0);}
            } catch (Exception e) {}
        if(messagesno == 1)
            {
                message = st.nextToken();
                System.out.println("current message sent is "+
message);
                try{
                    out.writeBytes(message + '\n');
                }catch(Exception e){}
                end = false;
                current_state = targetState(current_state,
```
(Continued) Client_simulator.java:

```java
System.out.println("current state is "+current_state);
if(index(EndStates,current_state) != -1)
    try{
        client.close();
        System.out.println("End state is reached");
        System.exit(0);
    }
    catch(Exception e){}

if(messagesno > 1)
{
    random = r.nextInt(messagesno)+ 1;
    for(i=0;i<random;i++)
    {
        message = st.nextToken();
        System.out.println("current message sent is "+message);
    }
    try{
        out.writeBytes(message +\n        );
    }
    catch(Exception e){}
    end = false;
    current_state = targetState(current_state,
    message);
}
System.out.println("current state is "+current_state);
if(index(EndStates,current_state) != -1)
    try{
        client.close();
        System.out.println("End state is reached");
        System.exit(0);
    }
    catch(Exception e){}
}//end if hasTemporal
}
//end while !end
//end run()
}
//end RemindTask

tic = new TemporalInteractionContract();
tic = tic.startParsing("C:/Documents and Settings/Einstein/Desktop/UniFrame/"+args[0]+".xml");
System.out.println("Starting the simulation for "+args[0]+" component");

System.out.println("====================================

States = tic.getStates();
Start = tic.getStartState();
Messages = tic.getMessages();
Transitions = tic.getTransitions();
EndStates = tic.getEndStates();
```
(Continued) Client_simulator.java:

```java
client = new Socket("localhost", 2000);
out = new DataOutputStream(client.getOutputStream());
System.out.println("current state is "+ current_state);

if(index(EndStates, current_state) != -1)
{try{client.close();System.out.println("End state is reached");System.exit(0);}catch(Exception e){}}

while(!end)
{
    if(hasTemporal(current_state))
    {
        if(use_rand) random = r.nextInt(randomfac);
        else random = 1;
        duration = temporalValue(current_state);
        if(random == 0)
        {
            temporal_message = temporalMessage(current_state);
            next_state = targetState(current_state, temporal_message);
            System.out.println("Duration is" + duration);
            timer = new Timer();
            timer.schedule(new RemindTask(next_state), duration*1000);
            end = true;
        }//end if random == 0
        if(random > 0)
        {
            messages = transitionMessages(current_state);
            StringTokenizer st = new StringTokenizer(messages, "$");
            messagesno = st.countTokens();
            if(messagesno == 0) client.close();
            if(messagesno == 1)
            {
                message = st.nextToken();
                System.out.println("current message sent is "+ message);
                timer = new Timer();
                timer.schedule(new RemindTask(current_state, message,
true), (duration-1)*1000);
                end = true;
            }
        }
    }
}
```
(Continued) Client_simulator.java:

```java
if(messagesno > 1)
{
    random = r.nextInt(messagesno) + 1;
    for(i=0;i<random;i++)
    {
        message = st.nextToken();
    }
    System.out.println("current message sent is "+message);
    timer = new Timer();
    timer.schedule(new RemindTask(current_state, message,
    true), (duration-1)*1000);
    end = true;
}
//end if random=1
} //end if hasTemporal
if(!hasTemporal(current_state))
{
    end = false;
    messages = transitionMessages(current_state);
    StringTokenizer st = new StringTokenizer(messages, ";");
    messagesno = st.countTokens();
    if(messagesno == 0) client.close();
    if(messagesno == 1)
    {
        message = st.nextToken();
        System.out.println("current message sent is "+
        message);
        out.writeBytes(message + \\
        System.out.println("current state is "+current_state);
        if(index(EndStates,current_state) != -1)
        {
            try{client.close();System.out.println("End state is reached");System.exit(0);}catch(Exception e){}}
        }
    }
}
```
(Continued) Client_simulator.java:

```java
if(messagesno > 1)
{
    random = r.nextInt(messagesno) + 1;
    for(i=0;i<random;i++)
    {
        message = st.nextToken();
        System.out.println("current message sent is "+message);
        out.writeBytes(message + \n);
    }
    System.out.println("current state is "+current_state);
    if(index(EndStates,current_state) != -1)
    {
        try{client.close();System.out.println("End state is reached");System.exit(0);}catch(Exception e){}}
}
//end if hasTemporal
//end while !end

}//end main
```