Viewpoints consistency management using belief merging operators

Qiuming Lin
University of Wollongong

VIEWPOINTS CONSISTENCY MANAGEMENT USING BELIEF MERGING OPERATORS

A thesis submitted in fulfilment of the
requirements for the award of the degree

MASTER OF INFORMATION SYSTEMS BY RESEARCH

from

UNIVERSITY OF WOLLONGONG

by

QIUMING LIN

School of Economcs & Information Systems

2004

CERTIFICATION

I, Qiuming Lin, declare that this thesis, submitted in fulfilment of the requirements for the award of Master of Information Systems by Research, in the School of Economcs & Information Systems, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Qiuming Lin

March 2004

Table of Contents

Abstract

Handling inconsistent requirement specifications is a critical and difficult issue in requirements engineering. There has been considerable research interest in this topic and many methods have been proposed and implemented in the past. This research aims at developing an approach to viewpoint merging for inconsistent management. The recent literature on belief merging provides several well defined merging operators that can be useful for viewpoints merging. This research has implemented a system for merging viewpoints specified as finite state models, in order to demonstrate that belief merging operators can indeed be the basis for viewpoints merging. We extend the state of the art by providing a technique for incremental viewpoints elicitation, and by addressing the problem of iterative merging in the present of viewpoints.

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Aditya Ghose for his many insightful comments and thoughts that guided me to finish this research. I am also thankful to Dr. Thomas Meyer for his work and helping me to understand his merging operator formulas. Thanks also to my other colleagues in Decision Systems Laboratory (DSL) for their valuable comments, supports, helps and encouragement during the process of completing this thesis as well as during the period of my master study.

Chapter 1

Introduction

Software tools make it much easier to manage the complexities of a system and help assure the construction of high-quality systems within time and budget limits. Requirements engineering has proven to be one of the most problematic areas within software development. Reconciling the requirements of multiple and disparate stakeholders involves effort on the part of system analysts, adding to the cost/time of system construction. We address this problem in this dissertation, by adapting and augmenting a formal framework for belief merging and by building a inconsistency management tool based on this framework, in the specific instance where requirements are specified as finite state machines.

1.1 Motivation

Requirements define the functionality and performance characteristics of systems to be implemented. These include descriptions of system behavior, application domain information, constraints on the system's operation, or specifications of system properties or attributes. Sometimes they are constraints on the development process of

the system. Requirements are further divided into functional requirements and non-functional requirements. The former are descriptions of what the system must do, while the latter are the properties or qualities such as operational cost, reliability, project budget etc. that a system must have, which are not directly related to functionality.

Requirements engineering is a branch of software engineering and can be defined as "the systematic process of developing requirements through an iterative cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained" [11]. The process of requirement engineering consists of such activities as requirements elicitation, requirements analysis and negotiation, and requirements validation [46].

Requirements engineering is the first stage of the software development life cycle. As part of this stage, requirements are discovered through consultation with stakeholders. Stakeholders are people who will be affected by the system and who have a direct or indirect influence on the system requirements [46]. They can be end users, clients, customers, analysts, managers, etc. Hence stakeholders may have different perspectives on the system as they may have different interests in the system and express their requirements in different vocabularies. Inconsistency among these requirements is therefore inevitable.

Handling inconsistent requirements is very important, as inconsistent requirements will have adverse impact on the system development. Requirements specifications that contradict with each other or system constraints often impede the process of system development. Worse still, the inconsistencies may be neglected by the analysts or

hidden in the documents and found at a later stage, in which case the options are either to roll back the entire development process to the early phases of the life-cycle or to perform imperfect patchwork repairs. In either case, the costs are likely to be unacceptably high. Therefore, inconsistency management is a critical but difficult task in requirements engineering.

There has been considerable recent research interest in this topic and several different methods have been proposed to handle the problem of requirement inconsistency. Generally, there are two approaches towards inconsistency handling. One is to enforce consistency so that inconsistency is not allowed (see [49] for example). The second involves inconsistency management that permits the existence of inconsistencies in a specification and supports a principled yet pragmatic approach to resolving these inconsistencies when appropriate (see [20] for example).

Ultimately, the challenge in multi-perspective requirements engineering is to reconcile the multiple, possibly inconsistent perspectives to obtain a single consistent specification. Inconsistency handling is clearly a key element of this process. Traditionally, this problem has been addressed by the process of requirements negotiation. The WinWin negotiation model based on Theory W [5] is one of the most commonly cited approaches to this problem. In this model, the stakeholders first provide their requirements, which are called Win Conditions. If a conflict is identified, an Issue schema, which contains the conflict, the Win Conditions affected and the stakeholders involved, is formed. The stakeholders then prepare their Options (alternative solutions) addressing the Issue for evaluation among the stakeholders until a satisfactory Option is achieved through an Agreement schema. More details on this will be presented in Section 2.2.

Our problem has much in common with the problem addressed in social choice theory by Kenneth J. Arrow (1972 Nobel Prize Winner for economics) in 1951 [1]. Social choice theory aims at providing a fair and equitable aggregation procedure for combining individual preferences into a new single social preference relation. Similarly, belief merging concerns constructing operations for merging the preferences of the individuals into a new single preference, as well as merging a finite set of knowledge bases that represent the beliefs of an agent, into an appropriate consistent knowledge base. Konieczny and Pino-Perez [27] have proposed a merging operation and defined a set of properties the merging operation should satisfy. There has been much earlier work on merging, notable examples being [3, 48, 31, 40, 30]. This research takes the work on belief merging as its point of departure. Within the framework of belief merging proposed by Meyer [33], we define an incremental elicitation approach to merging inconsistent viewpoints specified as finite state machines. We identify a class of merging operators, from the repertoire defined by Meyer [33], that are applicable in the context of incremental elicitation of progressively relaxed viewpoints. We present a tool to support viewpoint merging based on this approach, and we present some experimental results and case studies to validate this approach.

1.2 Main Contributions

The main contribution of this thesis is to argue, and in part, establish that belief merging operations are useful in reconciling multiple stakeholder perspectives in requirements engineering. We prove this by developing a framework implementing merging operations to handle multiple/inconsistent requirement perspectives represented as finite state models and propose an interactive, demand-driven model algorithm for

requirements elicitation.

The merging operations applied are based on the idea of Meyer's merging operations [33] defined in terms of epistemic states. A epistemic state provides a set of preference rankings of the models. The lower the ranking the more preferred it is deemed to be. In our framework, we have defined a ranked structure which has the same structure as the epistemic state. But it does not provide a full ranking of preference, it only provides preference rankings to a set of elicited models.

Requirements involved in our framework are represented as finite state models (FSMs), which have been used in many contexts, both conceptual and technical. FSMs are useful also because they are simple to understand, simple to reason about and simple to build. FSMs are especially interesting because they can be used in conjunction with a class of efficient model-checking procedures for verifying that the specified models satisfy relevant system properties (these may include liveness, fairness and safety properties).

Our framework contains the following components.

- A set of ranked structures (these may be viewed as partially specified epistemic states in the sense of Meyer's framework), each representing the viewpoint of a given stakeholder.
- A consistency checking device, which handles different vocabularies across the viewpoint models and perform consistency checking among the viewpoint models.
- A combination device to combine the consistent models into one model.

- A set of merging operators to merge the preference ranks of consistent sets of models into a new single list of preferences.
- SMV model checker [32] for verifying the system properties of the combined model.

A viewpoint in our system is assumed to be represented as a finite state model. First, the stakeholders provide their partial ranked structures with their most preferred models, which are checked by the tool for agreement, i.e. whether there are identical or consistent models among all the stakeholders. If no agreement is reached, the stakeholders are required to present their next most preferred models by either modifying the existing models or making new ones (and the models are added to their ranked structure), until an agreement is reached. Once an agreement is reached, the consistent models are combined into a single model with a merging operator selected to determine the new preference ranking for the combined model, hence creating a new and merged ranked structure containing the combined model. The combined model of the most preferred ranking is checked using the SMV model checker (more information on it is presented in chapter 2) to check against the system properties provided by the stakeholders (we do not address the problem of possible inconsistencies within these, and assume them to be consistent. If it does not satisfy all the properties, models of next preferred level in the merged ranked structure is model checked against the system properties until a satisfactory model is found. The model satisfying all of the system properties as well as the newly created ranked structure are the final outcomes of the tool. In the case where none of the models in the newly created ranked structure satisfy the properties, we have to continue the process by asking

the stakeholders to provide their models of the rank higher than the one where they reached the agreement, and the same process is iterated until a satisfactory model is found.

We describe the design and implementation of a tool that implements the scheme described above. We also present some case studies where this tool is deployed in semi-realistic situations.

1.3 Organization of the Thesis

The thesis consists of four chapters. Chapter 2 provides a brief background introduction to the areas related to our framework. First is the literature review of inconsistency management in requirements engineering. Inconsistency is explained with examples from [20]. Inconsistency management strategies and approaches to it are discussed. Then we briefly introduce other related areas, including requirement negotiation, social choice theory, belief merging and finite state model formally representing the specifications. In this chapter we also discuss the work of Chechik and Easterbrook that is closely related to what we are doing.

In Chapter 3, we present more details of the approach developed in this research. We first discuss the epistemic state, properties for combining epistemic states, a ranked structure defined based on the idea of epistemic state and the merging operators used. The SMV model checker incorporated into our framework is also briefly introduced. Then we explain how to merge multiple viewpoints and present the algorithm for constructing the framework. An example is presented in the last section to further explain how the framework works.

We discuss the implementation in Chapter 4, where we describe our system design and demonstrate how the prototype works with the major screen shots provided. The last chapter concludes the thesis by pointing out the limitation of our framework and the future work.

Chapter 2

Background

In this chapter, a brief literature review of the major areas related to our framework will be presented. Those areas are inconsistency management in requirements engineering, requirement negotiation, belief merging and formal specifications via finite state models. Finally the framework to merging inconsistent requirements proposed by Easterbrook and Chechik is discussed.

2.1 Inconsistency Management in requirements engineering

Inconsistency may arise due to mistakes, misunderstandings, or lack of information. It may also be the result of impractical requirements or conflicts between different perspectives. In the following, we briefly discuss inconsistency and inconsistency management strategies, and then a range of approaches to inconsistency handling is presented.

2.1.1 What is Inconsistency?

Inconsistency is generally defined as *any situation in which two descriptions do not obey some relationship that should hold between them* [36]. Inconsistency is inevitable in requirements engineering usually for the following reasons:

1) Inconsistent requirements specifications: Due to mistakes, misunderstandings, or lack of information, inconsistencies may occur in the requirements specifications. These are called local inconsistencies and are usually associated with a single stakeholder.

2) Inconsistent perspectives among multiple stakeholders: Inconsistencies arise more often among the requirements of multiple stakeholders. Different stakeholders may have their own knowledge, responsibilities, interests and commitments, so it is common to find that their requirements specifications contradicting each other. We call these global inconsistencies.

3) Requirement evolution: As the system environment as well as stakeholder requirements specifications change, new requirements may be added and existing ones may be deleted, hence causing contradiction among the requirements.

4) Inconsistency between functional requirements and non-functional requirements: Functional requirements may contradict non-functional requirements (a common example involves the competing pulls of speed and greater functionality).

Relationships between descriptions can be expressed as consistency rules, against which descriptions can be checked. A logical inconsistency is one instance of this

definition (in logical inconsistency, inconsistency occurs when both some fact A and its negation not A are derived), but other instances can also exist.

We shall consider the following examples from [20]. These examples are based on the TRMCS (Teleservices and Remote Medical Care System) case study used for IWSSD (International Workshop on Software Specification and Design). The examples are written in a language similar to KAOS (Knowledge Acquisition in autOmated Specification) language, and contain three parts. The first is the system requirements, which are also called goals. The second part is the formal definition of the requirement, represented in many-sorted first-order logic. The third part is further explanation of the requirement, written in English language.

In these examples, paramedical professionals (paramedics) and quality assurance (QA) professionals are stakeholders in the system. QA professionals require access to paramedic activity logs to better monitor their performance. Formally:

Goal 1 *Maintain* [QAAccessParamedicActivityLog]

FormalDef $\forall$ p: Paramedic, q: QAProfessional, l: ActivityLog

Records(p,l) $\rightarrow$ Accesss(q,l)

InformalDef: *Activity logs of every paramedic are accessible to all QA professionals.*

Goal 2 *Maintain* [ParamedicActivityLogAccess]

FormalDef $\forall$ p: Paramedic, q: QAProfessional, m: MedicalPractioner, l: ActivityLog

Records(p,l) $\rightarrow$ Accesss(m,l) $\wedge$ $\neg$Accesses(q,l)

InformalDef: *Activity logs of every paramedic are accessible to all medical professionals but not to any QA professional.*

It is obvious that if

∃ p: Paramedic, l: ActivityLog

Records(p,l)

holds in the system, then Goal 1 and Goad 2 of distinct stakeholders defined above are inconsistent. This is an example of a conflict between distinct stakeholder groups and between distinct functional requirements.

Goal 3 *Achieve* [DispatcherAccessPatientRecords]

FormalDef ∀ p:Patient, d: Dispatcher, r: PatientRecord, e: Event

Emergency(e,p) ∧ History(p,r) ∧ Manages(d,e) → AccessesDuringEvents(d,r,e)

InformalDef: *If a dispatcher is involved in the management of a medical emergency concerning a given patient, then the dispatcher has access to the medical history of that patient for the duration of the emergency.*

The rationale for this goal is another goal that requires that dispatchers be able to communicate relevant portion of a patient's medical history to paramedics during a medical emergency involving that patient.

Goal 4 *Achieve* [PatientRecCommunicatedParamedics]

FormalDef ∀ p: Patient, d: dispatcher, r: PatientRecord, e: event, m: Paramedic

Emergency(e,p) ∧ History(p,r) ∧ Manages(d,e) ∧ Responds(m,e) → Communicates-DuringEvent(d,m,r,e)

InformalDef: *Dispatchers managing a medical emergency involving a patient communicate that patient's medical history to paramedics responding to the emergency.*

Goal 5 *Maintain* [MobileAccessPatientRecords]

FormalDef ∀ c: MobileComputingDevice, r: PatientRecord DeviceAccess(c,r)

This is a different goal that requires mobile computing devices be equipped to directly access patient records from help center data servers.

Goal 6 *Achieve* [ParamedicAccessPatientRecords]

FormalDef $\forall$ p: Patient, r: PatientRecord, e: Event, m: Paramedic

Emergency(e,p) $\wedge$ History(p,r) $\wedge$ Responds(m,e) $\rightarrow$ AccessesDuringEvent(m,r,e)

InformalDef: *Paramedics responding to a medical emergency involving a patient are able to directly access that patient's medical history.*

The rationale for this is a goal that requires that paramedics be able to directly access a patient's medical records during an emergency involving that patient.

Goal 7 *Avoid* [RedundantAccess]

FormalDef $\forall$ x, y: HealthProfessional, r: PatientRecord, e: Event Communicate-DuirngEvent(x,y,r,e) $\wedge$ x $\neq$ y $\rightarrow$ $\neg$AccessesDuringEvent(y,r,e)

InformalDef: *If a patient history is communicated to a health professional y by another health professional x during an event, then y does not require direct access to the patient history during that event.*

Both dispatchers and paramedics belong to the HealthProfessional sort. If we know that states of the system exist where the following is true:

$\exists$ p: Patient, d: Dispatcher, r: PatientRecord, e: Event, m: Paramedic

Emergency(d,e) $\wedge$ History(p,r) $\wedge$ Manages(d,e) $\wedge$ Responds(m,e)

Then we are able to detect that Goal 3, Goal 5 and Goal 7 are jointly inconsistent, Goals 4, 6, 7 are also jointly inconsistent. These are examples of conflict between rationales.

Goal 8 *Maintain* [FastAccessPatientRecords]

FormalDef $\forall$ u: User, r: PatientRecord, t: TimeInterval

AccessDelay(u,r,t) $\rightarrow$ t $\leq$ 30

InformalDef: *The delay in accessing a patient record must be no more than 30 seconds.*

Goal 9 *Maintain* [SecureAccessPatientRecords]

FormalDef $\forall$ u: User, r: PatientRecord

AccessRequest(u,r) $\rightarrow$ Authenticate(u,r)

InformalDef: *If a user requests access to a patient record, then the system must authenticate that request.*

Given the domain theory:

$\forall$ u: User, r: PatientRecord, t: TimeInterval

Authenticate(u,r) $\wedge$ AccessDelay(u,r,t) $\rightarrow$ t $>$ 30

Goal 8 and Goal 9 are inconsistent, which is an example between a functional requirement and a non-functional requirement.

In the following, we will briefly describe the process of inconsistency management in requirements engineering.

2.1.2 Inconsistency Management

The process of inconsistency management generally consists of detecting inconsistency, diagnosing inconsistency, handling inconsistency and then monitoring the outcome. Consistency rules play an important role in inconsistency management. They are used to check the descriptions/specifications of requirements in order to identify

conflicts/contradictions. This set of rules will be improved and expanded as the inconsistency management cycle iterates [16]. Therefore, consistency rules must be expressed precisely.

The choice of inconsistency handling strategy depends on when it arises and how it impacts other aspects of the development process. Generally, there are two strategies. One is the conventional one that inconsistency is avoided, i.e. it will be rejected or resolved immediately after being detected. However, it is undesirable and costly to maintain consistency all the time through the software development process. Therefore, more and more researchers are advocating a different strategy, living with inconsistency.

Living with inconsistency allows the existence of inconsistency until it is resolved at the appropriate stage so that developers can continue their work without being constrained by the conflicts with others. It is most useful in handling inconsistency during specification evolution. According to [16], there are different actions that can be taken when an inconsistency is detected:

- Ignore - The presence of inconsistency can be ignored if the inconsistency does not have any significant impact on the development process. However, it is important to record and keep track of all inconsistencies even if they are ignored.
- Circumvent - Inconsistency is bypassed by modifying or disabling the rule for a specific context if a consistency rule is not applicable to that context or if the inconsistency represents an exception to the rule.
- Defer - Resolution of inconsistency is deferred and development can continue until it is deemed appropriate to handle the inconsistency.

- Ameliorate - Requirements causing inconsistencies can be repaired/modified instead of being deleted. This is only suitable if the modification of requirements does not have major side effects. The process is also referred to as incremental resolution.

In the next section, we are going to outline different approaches to inconsistency management.

2.1.3 Approaches to Inconsistency Management

Approaches to inconsistency management can be grouped into the following categories. Some of these approaches are based on similar ideas, but are implemented differently. For example, there are approaches that advocate formalization of the requirements specifications, but represent the specifications in different formal languages.

Ontological Approaches

This approach seeks to identify conflicts by providing a set of meaningful terms, or ontologies, by which one can specify conflict relationships between requirements. The idea is to explicitly state the vague and imprecise requirements. This can be useful for automated analysis of the specification.

[50] is an example of this approach that was proposed for analyzing the trade-offs between conflicting requirements. The approach employs utility functions from decision science [26]. These are used to validate the structure used in aggregating

prioritized requirements, to identify the structures and the parameters of the underlying representation of imprecise requirements and to assess the priorities of conflicting requirements.

Imprecise requirements are represented in fuzzy logic in this approach so that requirements can be described using linguistic terms, which make it easier to communicate and understand the requirements. An imprecise requirement can be satisfied to a degree and a satisfaction function, denoted as Sat_R, maps a requirement's domain to the range of satisfaction degree. Two imprecise requirements, R_1 and R_2 are said to be conflicting with each other if an increase in the degree of satisfaction of $R_1(R_2)$ often causes a decrease in the degree of satisfaction of $R_2(R_1)$. If an increase in the satisfaction degree of one requirement always decrease the satisfaction degree of the other, they are completely conflicting [50]. The approach also use conjunction and disjunction operators in fuzzy logic, where $\bigotimes$ is a fuzzy AND and $\bigoplus$ is a fuzzy OR.

[7] is another example of ontological approach, which adopts the Non-Functional Requirement (NFR) framework to deal with changes. It treats NFRs as goals to be achieved during the process of system evolution. Through the process, goals are decomposed, design tradeoffs are analyzed, design decisions are rationalized, and goals are evaluated.

The approach utilizes structures similar to semantic nets, in that it uses link types such as AND and OR links to connect the parent goals with the decomposed goals (offspring goals). Goals are captured and shown in a goal graph. Guidelines for changes are offered by use of the notion of structurally traceable goal graph to provide syntactic principles for maintaining the consistency of a goal graph.

The two approaches mentioned above suggest the need for an explicit semantics to bridge the gap between the imprecise requirements and formal specification method. The proposal in [50] is simple and explicit, while the notation in [7] is somewhat more complex.

State Machine Based Approaches

With this approach, requirement conflicts are identified by a specific technique, or automation. Automated technique helps to easily detect many classes of errors in requirements specification. [21] is an example of this approach. It proposes a formal analysis technique, called consistency checking, for automatic detection of errors in requirements specifications, which are expressed in Software Cost Reduction (SCR) tabular notation. Below is brief description of this approach.

It is based on the Four-Variable Model, which describes the required system behavior, as a set of mathematical relations on four sets of variables - monitored and controlled variables and input and output data items. A monitored variable represents an environmental quantity or variable that influences system behavior; a controlled variable is an environmental quantity that the system controls.

SCR has four other constructs: modes, terms, conditions and events. A *mode* class is a state machine, defined on the monitored variables, whose states are called system modes and whose transition is triggered by events. A *term* is an auxiliary function defined on input variables, modes, or other terms that help make the specification concise. A *condition* is a predicate defined on one or more system entities (a system entity is an input or output variable, mode, or term) at some point of time. An *event* occurs when any system entity changes value.

SCR tables contain condition tables, event tables, and mode transition tables. Each table describes a function to define an output variable, a term or a mode class.

An automated consistency checker is developed to check the specification for syntax and type correctness, coverage, determinism and other application-independent applications. As for the automated consistency checking, the consistency checker determines whether a logical expression is a tautology, by applying a tableaux-based decision procedure.

Approach based on Formal Logic

There have been several proposals of logical treatment of the inconsistency of software specification [22, 38, 19, 20, 43, 39, 10, 51].

Logic-based approaches contribute to inconsistency handling by providing techniques that assist analysis and reasoning in the presence of inconsistency. The use of logic provides a precise and unambiguous language to identify inconsistencies in evolving multi-perspective specifications. It also provides the means to address issues of inconsistency management in a generic way that is independent of any particular software engineering method or formalism [22]. In the following, we briefly address three categories of the approach: logical abduction, belief revision and argumentation view.

1) Logical Abduction

Abduction is one of the three fundamental modes of reasoning, in addition to deduction and induction. In artificial intelligence, abduction is generally accepted as the search for a set of hypotheses to achieve some given goals without causing

conflicts, when combined with a given theory [25].

[38] proposes logical treatment of inconsistency of software specifications, based on the abduction as formal reasoning. According to [38], abduction provide a formal technique for handling inconsistency by permitting incremental evolution of conflicting requirement specifications, as well as allowing implementation by employing existing tools for handling theory change. The specification and consistency rules are represented in *quasi-classical* (QC) logic [23], which allows continued reasoning in the presence of inconsistency. Classical abduction reasoning is adapted to handle inconsistent specifications in this logic. The abductive process is a backwards reasoning mechanism. If a literal α needs to be removed from a given QC specification in order to resolve the inconsistency, the abductive process backward from all the resolution steps that have lead to that literal. If it reaches some relevant literals that in the specification, then the identified literals become the abducible anti-explanation of that initial literal and will be deleted since QC logic is monotonic.

2) Belief Revision:

[19], [43] and [51] are similar in that they all present a logical framework to reasoning about requirements evolution based on the theory of belief revision. A sufficiently rich meta-level logic is used to formally and accurately capture intuitive aspects of handling incompleteness and inconsistency in requirements. Operators are used to map between theories of this meta-level logic to provide formal basis for the theory change component. This framework is based on the idea of belief revision that seek to find solutions through minimal changes to specification in the presence of inconsistency in the system development.

For example, [43] proposes a computational method for minimal revised logical specification. It utilizes an abductive procedure to compute a set of abducibles for minimal revised specification, in which abducibles are generated to get consistency for a given program. The mechanism can be used to compute deletion and addition of specification, which may be potential causes of inconsistency.

3) Argumentation View:

[39] presents a logical framework for reasoning about inconsistent requirements in the context of multi-viewpoint requirements engineering process. It proposes an argumentation view of multiple requirements, which can analyze the sources of inconsistencies.

In this approach, multi-viewpoints are represented as arguments. Arguments with no counterarguments represent the acceptable class of arguments and different meanings of "counterarguments" are used to derivate different classes of acceptable requirement. These arguments are characterized by order: from weakly confident to strongly confident (i.e. consistent). Inference rules are created for intra-viewpoints reasoning (concerned gradual reasoning) and inter-viewpoints reasoning (concerned safe reasoning). Reasoning is represented by the degree of confidence obtained from previous ordering over requirements.

Other Approaches to Tolerating Inconsistency

The followings are some of the typical notations of the strategy of managing inconsistency in the presence of inconsistency.

1) Living with Inconsistency:

Schwanke and Kaiser [44] suggest that during large systems development, programmers often circumvent strict consistency enforcement mechanism in order to get their jobs done. They propose and approach to "living with inconsistency" during development, and describe a configuration management (CONMAN) programming environment that helps programmers handling inconsistency by:

- identifying and tracking different kinds of inconsistencies (without requiring them to be removed),
- reducing the cost of restoring type safety after a change (using a technique called "smarter recompilation"), and
- protecting programmers from inconsistent code (by supplying debugging and testing tools with inconsistency information).

2) Tolerating Inconsistency:

It was first introduced by Robert Balzer [2]. In this approach, constraints are relaxed by treating inconsistencies as temporary exceptions, which will eventually be corrected. Before correction, the violated data are guarded by a technique called Pollution Marker. By use of these guards, specifications can be modified to avoid any inconsistencies completely or to tolerate them by adjusting their behavior, while specifications that are insensitive to violations of a specific constraint, need not be guarded by its Pollution Marker and will continue normal processing of the database, including the inconsistencies.

When the violation is eventually resolved and the data is no longer inconsistent, the guards are automatically removed indicating that no further resolution activity is required. Accessibility of this data to specification is restored.

3) Making inconsistency respectable:

Gabbay and Hunter propose "making inconsistency respectable" in that inconsistencies can be viewed as signals to external actions as well as signals to internal actions that activate ore deactivate other rules [29]. According to this approach, inconsistency is not necessary resolved by eradicating it, but by supplying rules that specify how to act in the presence of such inconsistency.

4) Lazy consistency:

This approach was proposed by Narayanaswamy and Goldman as the base for cooperative software development [35]. Its aim is to identify the technical basis to support the resolutions of cooperative software development (CSD) problems that may arise due to the distributed nature of the development process.

This approach favors software development architectures where the proposed changes (PCs) that are about to occur as well as changes that already occurred, are announced or broadcasted within the context of a larger transactional unit called evolution step, where all of the object-level changes are grouped together and handled coherently as a single logic unit. All the impacted stakeholders are notified of the PCs and they can review all the affected objects and explicitly express approval, rejection to or make modification to the PCs that adversely affect their objectives. Through such process, the step is gradually become consistent. Such an intra-step consistency is one aspect of lazy consistency.

The causal relationships between proposed changes are maintained so that stakeholder negotiations and other organizational protocols can be supported to resolve the collision and conflicts. It makes each step internally consistent and consistent

with regard to other volatile steps that might be pursued concurrently. The process of eventually resolving the intra-step and inter-step consistency within the step-based transaction model is the so-called lazy consistency.

We have discussed a range of approaches to managing inconsistency and presented them based on different dimensions. However, our concern is in belief merging, which suits handling inconsistency arising among multiple stakeholders. In the next section, we briefly describe belief merging and the related areas.

2.2 Requirements Negotiation

Requirements negotiation is an early phase in the system development process. After requirements have been acquired, analysis for conflicts/inconsistencies, overlaps and omissions is carried out, as well as negotiation with different stakeholders to reconcile and agree on these requirements.

There have been many models proposed to support requirements negotiation. WinWin negotiation model [29] is one of them. It is developed using Theory W [5] to generate the objectives, constraints, and alternative, as the goal of Theory W is to "Make everyone a winner". Figure 2.1 shows the WinWin negotiation model. It has four artifacts, *Win Conditions*, *Issues*, *Options* and *Agreements*. Win Conditions are the individual requirements; Stakeholders start by entering their Win Conditions. If a conflict among the stakeholders' Win Conditions is detected, an Issue is formed to address the conflict, contradictory Win Conditions and stakeholders involved. Then Options are proposed by the stakeholders to resolve the identified Issue. Stakeholders

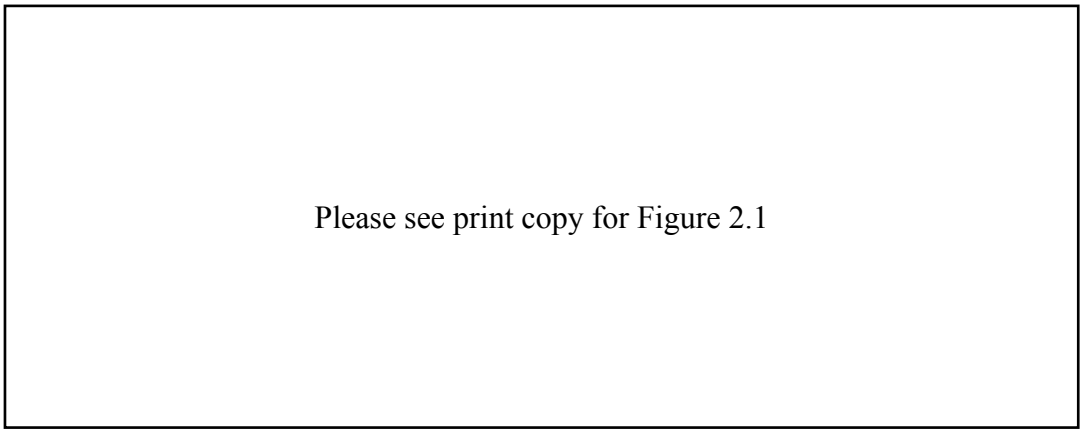

Figure 2.1: WinWin Negotiation Model [4]

evaluate, negotiate and reconcile on the Options and ultimately adopt a mutually satisfactory (i.e. win-win) one. The adoption of the final Option is called Agreement.

We shall explain how this negotiation model works more explicitly using the above hospital example. Assuming there are four stakeholders entering their Win Conditions which are Goal 5, 6, 8 and 9 (see section 2.1.1) respectively. Then the Win Conditions are examined to search for potential conflicts. In this example, conflicts arise between Goal 8 and Goal 9. Then stakeholders affected by these potential conflicts are identified, i.e. owners of Goal 8 and Goal 9, as well as a list of potential conflicts with the new Win Conditions are formed into an Issue artifact and provided to the stakeholders concerned (owners of Goal 8 and Goal 9 in this example), who will prepare their candidate Options addressing the issue. The Options are provided for stakeholders' evaluation, negotiation and adoption through the Agreement artifact.

Other similar approaches include Conflict-Oriented Requirements analysis [41, 42] (CORA) created to better analysis of the relationships among the system requirements. It provides requirement restructuring techniques to address issues arising through stakeholder analysis. It contains a cycle with three phases:

1) System requirements are defined.

2) Issues arise, through analysis of the requirements, which indicate possible conflicts among requirements.

3) Requirements are changed in response to conflicts.

The basic process of CORA is summarized as follows: System requirements are captured by defining the semantics of entities and relations. Then issues arise through analysis of requirements that may lead to varied stakeholder alternative solutions to the issues. In the conflict analysis process, a focusing strategy guides the order in which issues are to be analyzed for conflicts. A set of domain-independent transformations is provided to generate alternative requirements that remove stakeholder conflicts and the strategy for guiding the resolution generation process is defined. The transformations are changes to the meaning of requirements made by changing the classes or logical relationships specified in the requirements, rather than changes to design or implementation in order to satisfy requirements through various operationalization. A resolution is selected among a set of possible resolutions for each conflict. At last the requirement is updated with the newly defined requirement resolutions.

Viewpoints framework is another approach based on the concept of negotiation. There have been many works [16, 14, 13, 37, 12, 15, 17] approaching inconsistency management using Viewpoints framework, addressing the conflicts resulting from different perspectives of many stakeholders involved in the development of especially large and complex systems.

Viewpoints refer to the multiple perspectives that stakeholders maintain separately. In software terms, "Viewpoints are loosely coupled, locally managed, distributable objects that encapsulate partial knowledge about a system and its domain, specified in a particular, suitable representation scheme, and partial knowledge of the development process." [14] Each viewpoint consists of the following slots [17]:

- A representation *style*, the scheme and notation by which the Viewpoint expresses what it can see.
- A *domain*, which defines that part of the "world" delineated in the style.
- A *specification*, the statements expressed in the Viewpoint's style describing the domain.
- A *work plan*, which describe the process by which the specification can be built
- A *work record*, which contains an annotated history of actions performed on the Viewpoint.

In Viewpoints framework, inconsistencies between viewpoints are managed by explicitly defining relationships between them, and recording both resolved and unresolved inconsistencies. A viewpoint is locally consistent, while it may be inconsistent with other stakeholders'. Therefore, inconsistency is tolerated throughout the software development process.

Inconsistency checking is performed by applying consistency rules, which express the relationships that should hold between particular Viewpoints. When a consistency rule is applied, both the Viewpoints involved must collaborate to perform the

check and they both need to know the result. The Viewpoints might be evolving asynchronously and hence the application of the rule need to be performed as a single action.

Once an inconsistency is identified, (i.e. consistency check for relationship between two Viewpoints failed,) it will be corrected only if the owner wishes to do. Otherwise, inconsistency is tolerated. Resolution of inconsistency is re-establishing the relationship containing the rule that failed. If a relationship did previously hold, information about subsequent changes can be used to guide the resolution process. This information is available in the work record of each Viewpoint, as well as in the record of the results of previous consistency check.

2.3 Social Choice Theory

Social choice theory also offers a useful approach to reconciling the interest of different stakeholders' requirements. The theory is intensively studied in the fields of economics, political science and applied mathematics, and has lead to two Nobel Prizes in economics. It is most commonly applied in elections such as political elections or firm elections. It is also applied in the field of artificial intelligence to aggregate preference from different agents with different priorities, which is similar to our work in belief merging.

It is a theory of studying a decision among a collection of alternatives made by a group of n voters (truthful voters) with separate opinions. An individual voter's preference can be represented in total order $\leq$ over Ω. For example, for A, $B \in \Omega$, $A \leq B$ means A is at least as preferred as B. While the theory is interested in constructing

an aggregation operation on the preferences of n voters, i.e. $\leq_1,\ldots,\leq_n$ that generate a new preference ordering over Ω. Any choice for the entire group should reflect the desires of the individual voters to the greatest extent possible. In a social choice setting or voting, there will be different voting strategies. However, none of these strategies will guarantee consistency with Arrow's five desirable principles - the basis of his Impossibility Theorem. These principles are:

- *Universality*: The procedure should work for any preference configuration of individuals
- *Unanimity*: Unanimous preference of individuals should be respected. If all individuals prefer A to B then the social ranking will place A ahead of B.
- *Transitivity*: If the society prefers A to B and B to C, then the society prefers A to C.
- *Non-dictatorship*: The mechanism should not allow for dictator whose preference is dominant.
- *Independence of Irrelevant Alternative*: The relative ordering of two alternatives should be based only on their respective standings.

Arrow's work was later extended by Amatya Sen in many directions. Sen won Nobel Prize for economics in 1998 partly for his outstanding work in the field of social choice theory. One of Sen's major results dealt with the difficulties of reconciling libertarian ideals and considerations of efficiency.

Based on the ideal of liberalism, the individual is entirely decisive on any choice, no matter what the opinions the others have. If you want to drink water, others

should not force you to drink orange juice. While a mild liberal requirement would be that each individual should have a decisive voice over at least one alternative. An even weaker notion is that there should be at least two individuals who have a decisive voice over at least one pair of choice each. Sen called the latter minimal liberalism. On the hand, Pareto principle has long been regarded as an important requirement of "efficiency": for a pair of alternatives A and B, if everybody strictly prefers A to B, then B should not be chosen. A social decision function (the aggregation procedure) should also be universal and intransitive. Sen proved there couldn't be a social decision function that meets all these requirements simultaneously, which he called "the impossibility of a Paretian liberal".

2.4 Belief Merging

More and more research effort has been spent on belief merging over recent years and much exciting and interesting work has been done in this area. Belief merging is an approach to merging information from different sources and at different moments in time into a consistent one. It is based on the idea of belief revision, which happens when new information is added or removed from the agent's current set of belief. It is related to the AGM revision theory [9, 18], which proposed a set of postulates that any revising operator should satisfy.

There have been many different merging operations proposed in literature to resolve the inconsistency problem. Merging operation can be further divided into two sub operations, arbitration operation and majority operation.

Arbitration was first proposed by Revesz in [40] as a third type of theory change,

the others being revision and update. The idea of arbitration is that it is a possible situation where there is no reason to consider any of the different sources of information is more reliable than others. So the best outcome will be merging these views into a new and consistent one, while trying to retain as much information of different views as possible to maximize the individual satisfaction. In [40], Revesz introduces model-fitting and defines arbitration in the derived notion of model-fitting. He also discusses the properties an arbitration operator should satisfy. The approach proposed in [30] is similar to what Revesz proposes in that it merges different sources of information on the idea of arbitration as well as proposing the properties an arbitration operator should satisfy. But [30] also provides a set of postulates for arbitration and suggests to formalize arbitration by directly giving properties for it. [27] discusses some merging operators and also proposes a new arbitration operator and provides properties this arbitration operator should satisfy. All of this research relates to AGM (Alchóurron, Gärdenfors, Makinson) framework of revision theory [9].

Taking a different track, [31] proposes merging knowledge bases by majority. Here the viewpoints of the majority carry the most weight. There may be such a situation where "the number of agents who hold a particular belief is important, and sometimes it may be the only practical way of resolving a conflict." [31]. In [31], the properties that a majority operator should satisfy are proposed and principles of the majority operator are formalized.

[3] and [47] also discuss the issue of combining possibly inconsistent knowledge bases. But their goals and applications are different from the above discussed methods. In [47], Subrahmanian presents a framework to combine multiple knowledge bases based on *annotated logic*, a multi-value logic. In his work, he defines a model

where different databases are integrated via a supervisory database, which plays the role of *mediator* as referred by Silberschatz, Stonebraker and Ullman [45] to mediate between multiple knowledge bases. The supervisor mediates and chooses one piece of information over the other when conflicts arise. This approach best suits the situation where original databases cannot be modified.

Based on *logic program*, Baral *et al.* [3] tries to maximally combine multiple knowledge bases into a consistent combination with respect to the integrity constraints associated with the knowledge bases. It is possible that union of the knowledge bases can violate the integrity constraint, while each individual knowledge base does not. Even though the union itself is consistent, it may still violate the integrity constraints. So in case of contradiction, the knowledge base is converted into a disjunctive knowledge base to make it consistent. The users can choose among the disjunctive facts.

2.5 Formal Specifications via Finite State Models

Requirements can be represented in different ways. They can be in informal, semiformal or formal specifications. With informal specifications, there are no complete sets of rules to constrain the models that can be created. With semiformal specifications, syntax is defined. While for formal specifications, there are rigorously defined syntax and semantics, and a fundamental theoretical model against which a description can be verified.

With formal specification, the analyst is able to specify, develop, and verify a computer-based system by applying a strict and mathematical notation, allowing them to describe systems properties in a precise way. Therefore, formal specifications

help to reduce ambiguity, and improve consistency and completeness. They also help in verification of the specification and their implementation. There are many formal specification languages, such as Larch, which is an algebraic, sequential language. Z is a model oriented, sequential language and temporal logic is a model-oriented, concurrent language. The formal specification language we are concerned with here is finite state model (FSM), an abstract model of a system. It consists of the following components.

- A set of states
- A set of transitions between those states.

FSMs are widely applied in design and testing of automatic devices, telecommunication and computer hardware. They are also applied in software development, although to a lesser extent than in hardware development. In the last ten years, model checking, an automatic verification technique, has gained much attention in formal method application and FSMs are used to represent the finite-state concurrent system since the key property of a FSM is that all of the information about the process is captured in the current state. Its ability to represent an abstract model of a system is also useful in handling the "space explosion" problem in model checking.

2.6 Easterbrook and Chechik's Framework

Chechik and Easterbrook have proposed a merging approach to reasoning over inconsistent viewpoints [11]. It is based on multi-valued logic in which different values of the logics represent different levels of agreement.

In the approach proposed by Chechik and Easterbrook, logic is defined by using a lattice of truth values and the logical operators are defined in terms of lattice operations. It can ensure that a disjunction and conjunction of each pair of values exists and is unique in the logic by defining disjunction as join and conjunction as meet in the lattice. Besides disjunction and conjunction, it also can specify negation. Properties such as associativity, idempotency, distributivity, and De Morgan's laws hold in this logic and such logic is called *quasi-boolean* logics [6].

The viewpoint model used in the approach, or χview, is state machine model extended with a specific Quasi-Boolean logic in which Boolean variables can have a range of values, rather than just being "true" or "false". Now let us see how these χviews are combined into a merged model. Firstly, a signature map is defined to unify the vocabularies of the χviews as each χview is allowed to retain its local namespace in this approach. The signature map enforces that state names can only be mapped to state names and variable names mapped to variable names. Every state in the source χview must map to a state in the merged χview, but not necessary the variables. A name in a source χview may map to more than one name in the merged χview. Two different names from the same source χview cannot be mapped to the same name in the merged χview. Secondly, a value map is defined to map the truth values in the source χview to the truth values in the merged χview. After the merging, the merged χview is checked and analyzed using the symbolic multi-valued model checker χchek, which was developed by Chechik and Easterbrook.

The approach discussed here has advantages over the one with classical logic in that reasoning based on classical logic cannot reason about inconsistent and incomplete model because a single contradiction will result in trivialization. Properties of

individual viewpoints cannot be reasoned since these properties may change depending on how these viewpoints are combined.

2.7 Summary

In this chapter we have outlined several existing approaches to dealing with inconsistency in requirements engineering. These include a variety of approaches that apply to informal specifications as well as approaches that relate to formal specifications. We have also surveyed relevant literature in social choice theory and belief merging. We have discussed in the penultimate section an approach to combining specifications represented as finite state models. This approach is important but has several shortcomings, which have motivated our efforts to define an improved framework.

Chapter 3

Merging Viewpoints via Incrementally Elicited Ranked Structure

3.1 Preliminaries

3.1.1 The χbel framework

Our work takes as its starting point Chechik and Easterbrook's framework [11] for merging inconsistent viewpoints. The crux of their proposal is the use of multi-valued logic in which different values of the logic represent different levels of agreement. Thus, while individual viewpoints might be quite categorical in their specification, using only the TRUE and FALSE values (although the flexibility to use other truth values exists), the result may involve truth values other than TRUE and FALSE. Easterbrook and Chechik base their framework on a class of multi-values logics called *quasi-boolean logics*. Such logics are based on lattice of truth values with the logical operators defined in terms of lattice operations. Disjunction and conjunction of each pair of values exist and are unique as a consequence of disjunction being defined as

join and conjunction as meet in the lattice (these operations are also commutative, associative and idempotent in the lattice). Negation is also well-defined - the negation of each lattice element is another lattice element such that $\neg\neg a \equiv a$ for each lattice element a.

The notion of a viewpoint specified as a state transition model is extended in this approach in two ways. Variables which would be treated as boolean in conventional viewpoints are permitted to range over the set of truth values in the selected quasi-boolean logic. Viewing transitions as predicates (in conventional viewpoints, these would assume the value of TRUE if a transition existed between two states and FALSE otherwise), these too are allowed to range over set of truth values of the selected quasi-boolean logic. Such an augmented viewpoint is called a χview, and is defined as 6-tuple (L, S, S_0, R, I, A) where:

- L is a quasi-boolean logic with $\mathcal{L}$ denoting its set of truth values,
- S is a set of states, each with a unique label,
- $S_0 \subseteq S$ is a non-empty set of initial states,
- $R : S \times S \rightarrow \mathcal{L}$ is a total function assigning a truth value in $\mathcal{L}$ to each possible transition between pairs of states (including reflexive transitions). Each state is obliged to have at least one non-FALSE transition out of it.
- A is a set of atomic propositions, or variables,
- $I : A \times S \rightarrow \mathcal{L}$ is a total function giving truth values to each variable in each state.

Observe that each χview comes with its own set of variables and state labels as well as its own quasi-boolean logic. Since the task of unifying these across distinct χviews is difficult to automate, it is assumed to be a manual process executed by an analyst. Two distinct data structures are used to define the unification. A *signature map* unifies the vocabularies of a set of distinct χviews by determining which names in distinct χviews are synonyms and what the resulting name would be in the merged χview. A *value map* is a total function that maps each tuple of truth values in the quasi-boolean logics of the source χviews being merged into a truth value in the quasi-boolean logic underlying the merged χview (note that this assumes that such a logic has been selected).

The Easterbrook and Chechik approach is clearly an improvement over approaches that are based in one way or another on classical logic, since it avoids the problem of trivialization in the face of inconsistency. However, the approach is primarily useful for identifying sources of disagreement during viewpoint merging, rather than generating a merged viewpoint. For example, if the quasi-boolean logic underlying the merged χview is 4-valued, with values TT, FF, TF and FT, then variables and transitions having the value TT can be deemed to be fully agreed to by stakeholders as true while variables and transitions having the value FF can be deemed to be fully agreed to by stakeholders as false. But the variables with values of TF or FT are undefined and negotiation is necessary among the stakeholders to achieve a specific value. The value of the Easterbrook and Chechik framework lies in focusing attention on these areas of disagreement.

We note that a relatively straightforward approach based on maximal consistent

subsets provides a significant amount of guidance in generating specific merged viewpoint outcomes. The following definitions and the subsequent example illustrate this.

Definition 3.1.1. T_m, the syntactic representation, of a state transition model m, is defined as follows:

- For every state variable x which is assigned a value of TRUE in a state s of model m, holds(x, s)$\in T_m$
- For every state variable x which is assigned a value of FALSE in a state s of model m, $\neg$holds(x, s)$\in T_m$
- For every pair of states s_i and s_j in m where there exists a transition between s_i and s_j, trans(s_i, s_j)$\in T_m$. For all other pairs s_k and s_l in m, $\neg$trans(s_k, s_l)$\in T_m$.

Definition 3.1.2. A maximal consistent subset MaxCons(T_m) of the theory T_m that is the syntactic representation of a model m is defined as follows:

- $MaxCons(T_m) \subseteq T_m$
- $MaxCons(T_m) \nvDash \perp$
- For every T' s.t. $MaxCons(T_m) \subset T' \subseteq T_m$, $T' \models \perp$.

In general, several such maximal consistent subsets might exist.

Consider the setting shown in Figure 3.1(adapted from [11]), where two stakeholders(users) Alice and Bob present two distinct viewpoints, U_1 and U_2 respectively. For simplicity, we shall refer to these models using the names of the stakeholders that specified them. Using the syntactic approach based on maximal consistent subsets

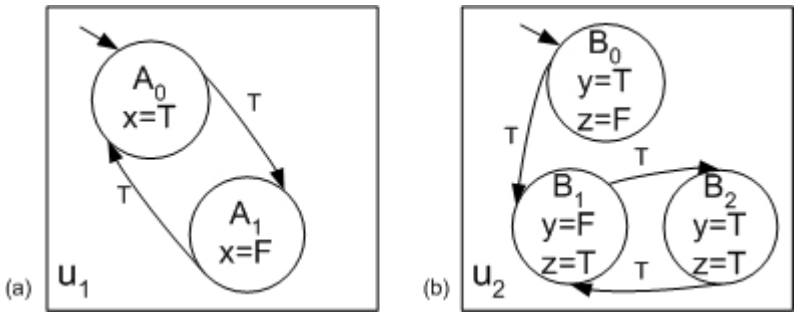

Figure 3.1: Sample Viewpoints

outlined above, we obtain the following. We assume here the existence of an analyst-generated signature map (see Table 3.1) which identifies the state labelled A_0 in U_1 with B_0 in U_2 and renames it as C_0 in the merged model (A_1 and B_2 are similarly identified and renamed as C_1, while B_1 is renamed as C_2). x maps to z and they are renamed as a, while y remains unchanged.

T_{U_1}={holds(a, C_0), ¬holds(a, C_1), trans(C_0,C_1) trans(C_1,C_0)}

T_{U_2}={holds(y, C_0), ¬holds(a, C_0), ¬holds(y, C_2), holds(a, C_2), holds(y, C_1), holds(a, C_1), trans(C_0, C_2), trans(C_2, C_1), trans(C_1, C_2), ¬trans(C_2, C_0), ¬trans(C_0, C_1), ¬trans(C_1, C_0)}

Several maximal consistent subsets of T_{U_1} and T_{U_2} exist and the followings are

Table 3.1: Signature Map

U_1	U_2	Mapping
A_0	B_0	C_0
A_1	B_2	C_1
	B_1	C_2
x	z	a
-	y	y

just two examples. We create the first maximal consistent subsets by taking Alice's viewpoint and adding as much of Bob's viewpoint as we consistently can. See Figure 3.2(a) for the combined viewpoint.

{holds(a, c_0), ¬holds(a, c_1), trans(C_0, C_1), trans(C_1, C_0), holds(y, C_0), ¬holds(y, C_2), holds(a, C_2), holds(y, C_1), trans(C_0, C_2), trans(C2, C_1), trans(C_1, C_2), ¬trans(C_2, C_0)}

The following maximal consistent subsets are created by taking Bob's viewpoint and adding Alice's viewpoint as much as we consistently can. See figure 3.2(b) for the combined viewpoint.

{holds(y, C_0), ¬holds(a, C_0), ¬holds(y, C_2), holds(a, C_2), holds(y, C_1), holds(a, C_1), trans(C_0, C_2), trans(C_2, C_1), trans(C_1, C_2), ¬trans(C_2, C_0), ¬trans(C_0, C_1), ¬trans(C_1, C_0)}

As we can see, the combined viewpoints obtained using the maximal consistent subset approach differ from the one (see figure 3.2(c)) obtained using the multi-valued logic approach proposed by Chechik and Easterbrook. Resolutions of inconsistency based on the maximal consistent subset approach provide more specific models as outcome, but require choice amongst multiple possible potential outcomes.

In this chapter, we present an approach to viewpoint merging using belief merging operators. We demonstrate this by implementing, as with Chechik and Easterbrook, a system for merging viewpoints specified as finite state models. We extend the state of art by providing a technique for incremental viewpoints elicitation, and by addressing the problem of iterative merging in the present of viewpoints.

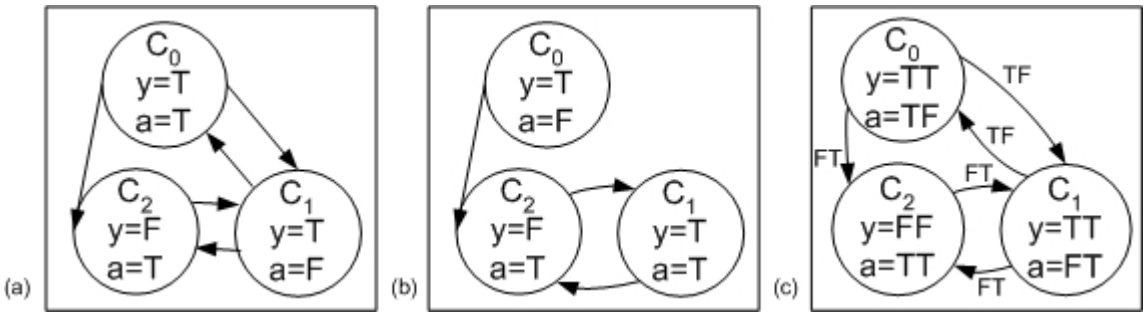

Figure 3.2: Merged Outcomes

3.2 Belief Merging

We base our work on Meyer's approach to belief merging [33]. We shall introduce this approach in this section. We begin by discussing the representation of beliefs using epistemic states. We then discuss some rationality postulates for belief merging operators. We then present a subset of Meyer's repertoire of merging operators.

3.2.1 Epistemic States

We shall assume a propositional language L, U is the set of interpretations of L and $M(\alpha)$ is the set of models of $\alpha \in L$. We shall use Φ to denote an epistemic state and ϕ to denote the knowledge base associated with Φ. We let x_n denote the list containing n version of x. The length of a list l is denoted by $\mid l \mid$.

In this and following subsections, we summarize several results from [33] that are relevant to our discussion.

Definition 3.2.1. An epistemic state Φ is a function from U to the set of natural numbers. Given an epistemic state Φ, the knowledge base associated with Φ, denoted by ϕ_Φ, is some $\phi \in L$ such that $M(\phi) = \{u \mid \Phi(u) = 0\}$.

Epistemic states allow us to represent preference orderings on valuation (or models). Valuations which receive a rank of 0 are the most preferred, while those that get a rank of 1 are the next most preferred, and so on. Some numbers may have no valuations assigned to them (i.e. there may be empty ranks) suggesting that the relative distance between ranks can play a role in the specification of preference.

An *epistemic list* $E = [\Phi_1^E, \ldots, \Phi_{|E|}^E]$ is a non-empty finite list of epistemic states. Each element of an epistemic list is an epistemic state representing the beliefs of an agent in the collection of agents whose beliefs must be merged. For any epistemic state Φ, let

$$min(\Phi) = min\{\Phi(u) \mid u \in U\},$$

let

$$max(\Phi) = max\{\Phi(u) \mid u \in U\},$$

and for an epistemic list E, let

$$max(E) = max\{max(\Phi_i^E) \mid l \leq i \leq |E|\}.$$

For an epistemic list E and $u \in U$, let $\min^E(\mathrm{u}) = \min\{\Phi_i^E(u) \mid 1 \leq i \leq |E|\}$ and let $\max^E(\mathrm{u}) = \max\{\Phi_i^E(u) \mid 1 \leq i \leq |E|\}$. $seq(E)$ denotes the set of all sequences of length $|E|$ of natural numbers, ranging from 0 to max(E). We denote by $seq \leq (E)$ the subset of $seq(E)$ of all sequences that are in non-decreasing order, and $seq \geq (E)$ the subset of $seq(E)$ of all sequences that are in non-increasing order. For $u \in U$, we let $s^E(\mathrm{u})$ be the sequences containing the natural numbers $\Phi_1^E(u), \ldots, \Phi_{|E|}^E(u)$ in that order, we let $s_{\leq}^E(u)$ be the sequence $s^E(u)$ in non-decreasing order, and we let $s_{\geq}^E(u)$ be the sequence $s^E(u)$ in non-increasing order. Obviously $s^E(u) \in seq(E)$, $s_{\leq}^E(u) \in seq_{\leq}(E)$ and $s_{\geq}^E(u) \in seq_{\geq}(E)$. $s_i^E(u)$, $s_i^{(E,\leq)}$ and $s^{(E,\geq)}$ denote the i-th digit in $s^E(u)$,

$s^E_{\leq}(u)$ and $s^E_{\geq}(u)$ respectively. Given any set *seq* of finite sequences of natural numbers and a total preorder $\sqsubseteq$ on *seq*, we define the function $\Omega^{seq}_{\sqsubseteq} : seq \rightarrow \{0, \ldots, |seq| - 1\}$ by assigning consecutive natural numbers to the elements of *seq* in the order imposed by $\sqsubseteq$, starting by assigning 0 to the elements lowest down in $\sqsubseteq$.

3.2.2 Properties for Combining Epistemic States

A *merging operation* Δ *on epistemic states* is defined as a function from the set of all non-empty epistemic lists to the set of all epistemic states [33] . [33] also proposes (E0) to (E6), (Arb) and (Maj) postulates (as listed below) that a merging operator should satisfy. Such postulates are useful in defining a yardstick for defining the rationality/correctness of the merging operators under consideration. Rationality postulates have been used previously in non-monotonic reasoning [28] and belief revision [9] literature for motivating specific classes of inference and revision operations. Similar postulates for merging have been previously be proposed by [27] and [30].

(E0) $\Delta([\Phi])(u) = (\Phi)(u)$ - $\min(\Phi)$

(E1) $\exists u \; s.t. \; \Delta(E)(u) = 0$

(E2) $\Phi^E_i(u) = \Phi^E_j(u) \; \forall i, j \in \{1, \ldots, |E|\}$ and $s^E_{\leq}(u) s^E_{\leq}(v)$ implies that $\Delta(E)(u) < \Delta(E)(v)$

(E3) If $\Phi^E_i(u) \leq \Phi^E_i(v) \; \forall i \in \{1, \ldots, |E|\}$ then $\Delta(E)(u) \leq \Delta(E)(v)$

(E4) if $\Delta(E)(u) \leq \Delta(E)(v)$ then $\Phi^E_i(u) \leq \Phi^E_i(v)$ for some $i \in \{1, \ldots, |E|\}$

The above are the basic properties for merging of epistemic states. Let ε be a finite list of epistemic list $\varepsilon = [E_1, \ldots, E_{|\varepsilon|}]$, there are following two properties:

(E5) $\Delta(E_i)(u) \leq \Delta(E_i)(v)\ \forall i \in \{1, \ldots, |\varepsilon|\}$ implies that $\Delta(\bigsqcup_{i=1}^{|\varepsilon|} E_i)(u) \leq \Delta(\bigsqcup_{i=1}^{|\varepsilon|} E_i)(v)$

(E6) If $\Delta(\bigsqcup_{i=1}^{|\varepsilon|} E_i)(u) \leq \Delta(\bigsqcup_{i=1}^{|\varepsilon|} E_i)(v)$ then $\Delta(E_i)(u) \leq \Delta(E_i)(v)$ for some $i \in \{1, \ldots, |\varepsilon|\}$

The arbitration postulate and majority postulate can be generalized as follows:

(Arb) $\forall n\ \Delta(E \sqcup [\Phi])(u) \leq \Delta(E \sqcup [\Phi])(v)$ iff $\Delta(E \sqcup \Phi^n)(u) \leq \Delta(E \sqcup \Phi^n)(v)$

(Maj) $\exists n\ s.t.\ \forall u, v \in U, \Phi(u) \leq \Phi(v)$ if $\Delta(E \sqcup \Phi^n)(u) \leq \Delta(E \sqcup \Phi^n)(v)$

(E0) states that combination with a singleton list should produce no change. But if there is no model assigned the preference rank 0 after the merging operation, which indicates an inconsistency in the knowledge base, then we should perform normalization by subtracting the minimal level to make it consistent. Therefore (E0) restricts to epistemic states of consistent associated knowledge base. (E1) says there exists a model that is assigned rank 0 after the merging operation, which requires that the model obtained should be consistent. (E2) says model u that are agreed by all of the epistemic states in E, should be strictly more preferred than any model v which is regarded by every epistemic state to be at most as preferred as u, but less preferred than u by at least one of the epistemic states. (E3) states that if all epistemic states in E agree that u is at least as preferred as v, then it should be the case in the resulting epistemic state. (E4) shows that if a model u is regarded as at least as preferred as v after completion of the merging operation, so there has to be at least one epistemic state in E which regards u as at least as preferred as v.

(E2), (E3) and (E4) postulates are regarded as the most basic and important ones. A merging operator satisfying these postulates has a rational behavior concerning merging. (E5) and (E6) postulates are just generalization of (E3) and (E4) respectively. (Arb) postulate states the characteristics of an *arbitration* operation, while (Maj) postulate explains that adding enough epistemic states Φ to the epistemic list E results in a refined version of Φ when combining epistemic states. In the next section, we describe some of the merging operations that satisfy the above postulates.

3.2.3 Merging Operators

In the following, we will review some of Meyer's merging operators. In particular, we will review three specific operators: Δ_{min}, Δ_{max} and Δ_{Σ}. Meyer defines several others, but these three form a representative subset. Δ_{min} and Δ_{max} are examples of arbitration operators while Δ_{Σ} is an example of a majority operator.

There are two steps in the construction of each merging operation. The first step is to assign the rank (natural number) to each model (or valuation). After completing this step, if *none* of the models have been assigned a value 0, then the second step is to perform an appropriate uniform substraction of values, which is referred to as *normalization*. In cases where there are no models of rank 0 (suggesting that the agent't beliefs are inconsistent) we normalize by shifting all of the ranks down, while maintaining their relative order and distance, but ensuring that the set of models at rank 0 are non-empty.

Arbitration

We consider the idea of an arbitration operation in which we take as many different viewpoints as possible from all the stakeholders into account. We will discuss two

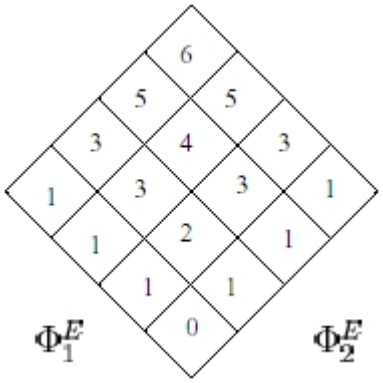

Figure 3.3: A representation of the merging operator Δ_{min}. The numbers in a cell represent the rank that the appropriate merging operation assigns to the viewpoints contained in that cell before normalization

arbitration operators, the first of which is the Δ_{min} merging operator.

Definition 3.2.2. If E contains a single epistemic state Φ, let $\Phi^E_{min} = \Phi$. If not, let $\Phi^E_{min}(u) = 2\min^E(u)$ if $\Phi^E_i(u) = \Phi^E_j(u)\ \forall i, j \in \{1, \ldots, |E|\}$ and $\Phi^E_{min}(u) = 2\min^E(u)$ + 1 otherwise. Then $\Delta_{min}(E)(u) = \Phi^E_{min}(u)$ - min(Φ^E_{min}).

Figure 3.3 [33] is a pictorial representation of merging operator Δ_{min}. The Δ_{min} operator involves the following steps. Identify the models which are agreed to by all epistemic states as being the most preferred, and take them to be the most preferred model in the resulting epistemic state from the merging operation (assigning them the rank of 0). The models on the next level of preference are those deemed to be the most preferred by at least one epistemic state. The models on the next level of preference are considered to be the ones that are deemed to be the second most preferred by all the epistemic states and the models regarded as the second most preferred by at least one epistemic state are on the following level of preference. The above process is repeated until all levels of preference for all the epistemic states have been treated. The idea of Δ_{min} is to find the minimum preferred rank given to a

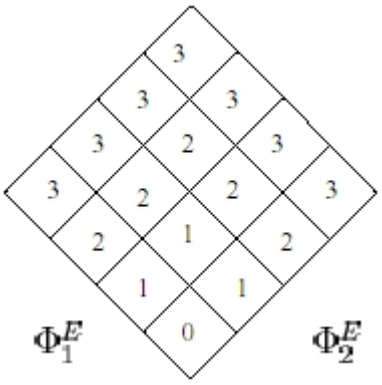

Figure 3.4: A representation of the merging operator Δ_{max}. The numbers in a cell represent the rank that the appropriate merging operation assigns to the viewpoints contained in that cell before normalization

model by any of the epistemic states and then to normalize the rank. The normalized rank is assigned as the new preference rank to the model.

Δ_{min} satisfies (E0)-(E5) and (Arb) properties (see [33] for proof), reflecting its suitability for the merging task. It does not satisfy (Maj) indicating that it is an arbitration operator rather than a majority operator.

Δ_{max} is another arbitration operator.

Definition 3.2.3. Let $\Phi^E_{max}(\mathrm{u}) = \max^E(\mathrm{u})$. Then $\Delta_{max}(E)(u) = \Phi^E_{max}(\mathrm{u})$ - $\min(\Phi^E_{max})$.

Figure 3.4 [33] represents Δ_{max}. The maximum preference rank assigned to a model by any of the epistemic states is taken as the preference rank to that model. It satisfies (E0)-(E6) and (Arb) properties (see [33] for proof) and it does not satisfy (Maj).

Majority

Majority operators take the viewpoints of the majority stakeholders into account, i.e. it tries to minimize global dissatisfaction. The Δ_Σ merging operation is an example of a majority operation. Before we come to the definition of the Δ_Σ operation, it is necessary to look at the following form of *summation* (which is used in defining Δ_Σ). For $s \in seq(E)$, let

$$sum^E(s) = \sum_{i=1}^{|E|} s_i$$

where s_i is the ith element of s.

Definition 3.2.4. Let $\Phi_\Sigma^E(u) = sum^E(s^E(u))$. Then $\Delta_\Sigma(E)(u) = \Phi_\Sigma^E(u) - min(\Phi_\Sigma^E)$.

As before, the final sentence in the definition represents the normalization step. Figure 3.5 [33] gives a pictorial representation of Δ_Σ. The idea of this operation is to obtain the new preference rank of the model by summing the preference ranks given by the different epistemic states (representing viewpoints of stakeholders) and then to normalize the ranks.

Δ_Σ satisfies (E0)-(E6) and (Maj), but it does not satisfy (Arb) (see [33] for proof), indicating it is a suitable majority operator.

There are also some other forms of combination, which are not given further description here, interested readers can refer to [33].

3.2.4 Model Checking Merged Viewpoints using SMV

A key element of our approach to merging multiple viewpoints is to generate *appropriated* merged outcome. When viewpoints are specified as state machines, the

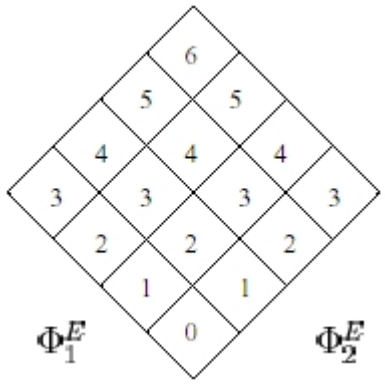

Figure 3.5: A representation of the merging operator Δ_Σ. The numbers in a cell represent the rank that the appropriate merging operation assigns to the viewpoints contained in that cell before normalization

satisfaction of a set of properties (including liveness, safety etc.), usually specified in a temporal logic, is an important concern. Our approach thus relies on the existence of some form of machinery to verify that a given model satisfies a set of properties. Since we restrict our attention to finite state models, a *model checker* (see [8] for a good introduction to model checking) is an obvious choice for this machinery.

We use the SMV ("symbolic model verifier") [32] model checker for analyzing the merged viewpoints obtained via our merging process. SMV is the best-known of model checkers that supports CTL (*Computation Tree Logic*). SMV takes two inputs, i.e. a model described in the SMV input language and some properties specified in CTL . It outputs either the word "true" if the specifications are satisfied in all initial states, or the word "false" with a counterexample showing why the specification does not hold in the model determined by the program.

CTL is a branching-time temporal logic. There are different paths in the future and

any of the paths may be the realized one. Basically, CTL has the following eight temporal connectives: **AX**, **EX**, **AG**, **EG**, **AU**, **EU**, **AF** and **EF**. **A** and **E** are the path quantifiers, i.e. **A** signifies "for all computation paths" and **E** represents "for some computation path" [8]. **X** ("next state"), **F** ("some future state"), **G** ("globally") and **U** ("until") are the basic temporal operators. These temporal operators must be preceded by one of the path quantifiers. The logic connectives $\vee$, $\wedge$ and $\neg$ are also used in CTL with the usual meaning. See [24] for more details on CTL.

The SMV model checker allows us to ensure that the model generated by our system satisfies the given properties. We do not address the problem of resolving inconsistencies amongst alternative sets of properties specified by different stakeholders. Several of the techniques discussed in the previous chapter (such as the framework presented in [20]) can be used for this purpose. We assume that there is a single consistent set of properties that all stakeholders agree to.

3.3 Merging via Ranked Structure

3.3.1 Ranked Structures

We introduce the idea of a *ranked structure* - a notion related to, but distinct from the notion of epistemic state used in Meyer's framework for belief merging. An epistemic state is intended to be a complete specification of an agent's epistemic state. Thus, it requires us to assign a rank to every possible state of affairs. In most non-trivial domains, the number of possible states of affairs is typically very large, and many of them, particularly at higher ranks (i.e. those that are less preferred), are often irrelevant to the discourse. In a realistic application domain, such as ours, we cannot conceivably have access to such a mapping. At best, we may ask agents (stakeholders)

to rank the models elicited thus far. A ranked structure can thus be loosely viewed as being analogous to a partially specified epistemic state. There is another critical difference. In Meyer's approach to belief merging, we assume a commonly agreed upon language, relative to which models (or states of affairs) are conceived. In our context, each viewpoint comes with its own local vocabulary, relative to which a stakeholder specifies models. A global (common) vocabulary is eventually constructed via signature maps (described below), but this results in individual stakeholder models becoming incomplete (in general) relative to this global vocabulary. This represents another point of departure from the notion of an epistemic state. Meyer, Ghose and Chopra [34] have defined a syntactic approach to merging using ranked knowledge bases, but these are expressively equivalent to epistemic states, and hence inapplicable in our context for precisely the same reasons as those listed above.

3.3.2 Signature Map

Given a set of viewpoints, although all of them are represented in finite state models (refer to Chapter 2 for more details), it is possible that they are expressed in different ways, using different vocabularies. Therefore, we require a *signature map* that unifies different vocabularies. We assume that the models do not share the same vocabularies, and the analyst can decide the matching values of the states and variables across the viewpoint models.

We follow the similar properties as in [11] to define our signature map.

- Type information preserved - state name can only be mapped to state names, and variable names can only be mapped to variable names.

- Every state and variable in the source models must map to a state and a variable in the merged model.
- Two different names from the same source model cannot be mapped to the same name in the merged model.

3.3.3 Guidelines for Selecting Merging Operators

Our framework provides a set of merging operators and the followings are the guidelines for choosing a merging operator.

Generally, there are two subclasses of merging operations as mentioned before. *Arbitration operators* try to take as many different opinions as possible into account to *minimize the individual dissatisfaction*. *Majority operators* take the viewpoints of the majority stakeholders into account, i.e. they try to *maximize global satisfaction*. Therefore, in cases where individual stakeholders are of similar importance, arbitration operators are the preferred choice. Majority operators, on the other hand, may produce outcomes that leave some stakeholders very happy but others that are not happy with the result at all.

Arbitration operators include such operators as Δ_{min} and Δ_{max}. With Δ_{min}, what effectively happens (with a minor variation) is that the lowest preference rank (indicating most preferred) assigned to a viewpoint model by at least one stakeholder will be taken as the preference rank for that viewpoint model in the resulting ranked structure. Δ_{max} takes the highest preference rank (indicating the least preferred) assigned to a viewpoint model by at least one stakeholder as the preference rank for that viewpoint model in the resulting epistemic state. Δ_{min} can be regarded as

an open minded operator in which if the viewpoint is highly regarded by at least one stakeholder, then the viewpoint is assigned the better preference rank. The Δ_{max} operator, on the other hand, is more conservative, as it assigns the poorest rank given by at least one stakeholder to the viewpoint.

Δ_{Σ} is a majority operator. Δ_{Σ} adds all the preference ranks assigned to a viewpoint by different stakeholders and takes it as the new preference rank to the model in the resulting ranked structure. Therefore, it takes all the stakeholders' viewpoints into consideration and tries to maximize the global satisfaction.

3.4 Algorithm for Merging via Incrementally Elicited Ranked Structures

In this section, we present an algorithm for constructing our framework. It is a *lazy valuation*, in that no full rank of epistemic states is necessarily provided. We just provide a ranked structure that contains the firstly given viewpoints models. The stakeholders start by giving their most preferred models, i.e models of preference rank 0 (It is possible that some stakeholders may have more than one model at a given rank in which case their models must be inconsistent). If the models presented by all the stakeholders are consistent, then we should retain all of these models and combine them into a single model. Otherwise, the stakeholders are required to supply their next most preferred models and the models are added to the ranked structure. They keep providing additional models until they reach an agreement, i.e. their models are identical or consistent. Once an agreement is reached, sets of agreed models are combined into single models with a merging operator selected to determine the new preference ranks for them, and a new and merged ranked structure is hence formed.

The merged ranked structure is comprised of only combined models and the preference ranks assigned to them. The most preferred models of the merged ranked structure is first taken to check against the system properties set by the stakeholders using SMV model checker. If SMV returns a true value, this model will be the result model. If SMV returns a false value, then models of the next preference level of the merged ranked structure are model checked until a model is found to satisfy the properties and such model is the final outcome model. If no such model is found in the merged ranked structure, then we have to keep asking the stakeholders to give their models of the next preference level from where they previously reached an agreement, and repeat the above process to find a successful outcomes. The following algorithm of procedure **IncrementalMerge**() reflects this process.

Not all operators in Meyer's repertoire of merging operators lend themselves to an incremental elicitation approach to merging. We define below the *incrementality property* to circumscribe the set of merging operators that do lend themselves to incremental elicitation.

Definition 3.4.1. A merging operator is said to satisfy the incrementality property iff it is able to generate a complete merged epistemic state, up to rank(r-1) if all epistemic states in input epistemic list are completely determined up to rank r, for $r \geq 1$.

It is easy to see that amongst Meyer's operators, only Δ_{min}, Δ_{max}, Δ_{Σ} and $\Delta_{R\Sigma}$ satisfy this property. In the rest of our discussion, we will only be interested in merging operators which satisfy the incrementality property.

r represents the rank of the ranked structure and i represents to the number of stakeholders through the algorithms described in this chapter.

---

procedure IncrementalMerge

---

inputs:

1. A set of partial ranked structures, $\{RS_i \mid i \in \text{STAKEHOLDERS}\}$
2. A merging operator OP and an associated function $\textbf{PartialMerge}_{OP}$
3. A set of CTL properties $PROP$

outputs:

1. A single partial ranked structure PM, represented as a sequence of sets $\langle\ S_0, S_1, \ldots, S_r\ \rangle$, organized in ascending order of rank, where each set S_i contain models at rank i.
2. A model m

$done$:= false

$r := 0$

repeat

for each stakeholder $i \in$ STAKEHOLDERS

elicit all models at rank r and place them in the set SM_r^i

$PM := \textbf{PartialMerge}_{OP}(\{SM_k^j \mid j \in \text{STAKEHOLDER}, k \in \{0, \ldots, \text{r}\}\})$

if the $(r-1)$th element of PM exists and is non-empty

if there is a model m in PM_{r-1} that satisfies all properties in $PROP$

$done$:= true

return m, PM

else

if there exists a model $m' \in PM_r$ that satisfies all properties in $PROP$

done := true

return $m^{'}$, *PM*

else

r := *r*+1

else

r := *r*+1

until *done*

We define **PartialMerge**$_{OP}$() for instances where the merging operations under consideration are Δ_{min}, Δ_{max} and Δ_{Σ} in the following discussion. Thus **PartialMerge**$_{\Delta_{min}}$ $(x)^{def}_{=}$ **PartialMerge**(x, min), **PartialMerge**$_{\Delta_{max}}$ $(x)^{def}_{=}$ **PartialMerge**(x, max) and **PartialMerge**$_{\Delta_{\Sigma}}$ $(x)^{def}_{=}$ **PartialMerge**(x, Σ) for all x. Other merging operations from [33] could also be supported by other instances of **PartialMerge**$_{OP}$(), but we do not elaborate them here, in the interests of brevity.

---

procedure PartialMerge()

---

inputs:

1. A set of partial ranked structures, one for each stakeholder. For a stakeholder i, a partial ranked structure is represented as sequence of sets $\langle\ SM^i_0, SM^i_1, \ldots, SM^i_j\ \rangle$ where each set SM^i_j contains the models specified by stakeholder i at rank j
2. A function *f*, determined by the merging operator *OP*

outputs:

1. A single merged partial ranked structure $S : \langle\ S_0, S_1, \ldots, S_k\ \rangle$

---

for each $m \in SM_j^i$ (for any i and any j)

CONS(m) = $\{\langle$n, j$\rangle \mid n \in \mathrm{SM}_j^i$, for any i and any j, s.t. **Consistent**$^*(m, n)\}$

$S_k := S_k \bigcup \{\langle n, k\rangle\}$ where $\langle n, k\rangle$ = **Combine**(CONS(m), f)

return S

Consistent$^*(m, n)$ is a test for the consistency of models m and n. Two models are consistent if the following rules are satisfied:

- If a variable is true in the state in one model, and the state is described in the second model, then the variable should be true or undefined in the state of the second model.
- If a variable is false in the state in one model, and the state is described in the second model, then the variable should be false or undefined in the state of the second model.
- If a transition between two states is described in one model, both of the states are described in the second model, then the transition should be described in the second model.

The following algorithm for function **Combine**() used in procedure **PartialMerge**() is for merging procedure involving the Δ_{max} and Δ_{Σ} operations.

---

function Combine(S,f)

---

inputs:

1. A set S of pairs of form $\langle m, l\rangle$, where m is a model and l is a rank such

that $l \in \{0, \ldots, r\}$. (All models referred to in S are guaranteed to be consistent).

2. A function f where $f = \max$ or $f = \Sigma$

outputs:

1. A combined model with its associated rank $\langle n, k \rangle$

$n := \{\}$

$k := 0$

for $l = 0, \ldots, r$ **do**

$n :=$ **CombineModels**$^*(n, m)$ where $\langle m, l \rangle \in S$

$$k := \begin{cases} \max(k, l) & \text{if } f = \max \\ k + l & \text{if } f = \Sigma \end{cases}$$

return $\langle n, k \rangle$

The following algorithm for function **Combine**() using the Δ_{min} is slightly different from the above.

---

function Combine(S, f)

---

inputs:

1. A set S of pairs of form $\langle m, l \rangle$, where m is a model and l is a rank such that $l \in \{0, \ldots, r\}$. (All models referred to in S are guaranteed to be consistent).

2. A function f where $f = min$

outputs:

1. A combined model with its associated rank $\langle n, k \rangle$

$n := \{\}$

rank-set $:= \{\}$

for $l = 0, \ldots, r$ **do**

$n :=$ **CombineModels**$^*(n, m)$ where $\langle m, l\rangle \in S$

rank-set $:=$ *rank-set* $\bigcup \{l\}$

$$k := \begin{cases} 2l & \text{if } \textit{rank-set} \text{ is a singleton and } \textit{rank-set} = \{l\} \\ 2\min(\textit{rank-set})+1 & \text{otherwise} \end{cases}$$

return $\langle n, k\rangle$

The function **CombineModels**$^*(m_1, m_2)$ takes two consistent models m_1 and m_2 as input and combine them into a single model m based on the following principles.

We use $Var_m(s_i)$ to denote the set of variables that are assigned a value in state s_i in model m. We note that in a completely specified model, $Var_m(s_i)$ should be identical for each state s_i, but we allow for the possibility that users may incompletely specify a model.

We use $\langle s_i, s_j\rangle$ to denote a transition from s_i to s_j

- If a state s_i is defined in both models m_1 and m_2, then s_i must be defined in m $Var_m(s_i) = Var_{m_1}(s_i) \bigcup Var_{m_2}(s_i)$
- If a state s_i is defined in model m_1 but not in the model m_2 (or the reverse, without loss of generality), then state s_i must be defined in model m. $Var_m(s_i)$ $= Var_{m_1}(s_i)$.
- If a transition $\langle s_i, s_j\rangle$ is defined in either m_1 or m_2, $\langle s_i, s_j\rangle$ remains a transition in m.

We have described thus far a procedure for merging the incrementally elicited viewpoints of a fixed set of stakeholders. In real-life applications, the set of stakeholders may change - new stakeholders may join and existing ones may have. In the following, we present an approach to deal with new stakeholders, i.e. an approach to iterated merging. The problem of altering the merged outcome to reflect that a given stakeholder's viewpoint is no longer applicable, without having to recompute from scratch, is a difficult problem (with similarities to the problem of belief contraction) and one that we do not address in this dissertation.

In the case where there are new stakeholders joining, then the new stakeholders first provide their models of rank 0, which will be compared with the combined models of rank 0 of the newly created merged partial ranked structure, which can be regarded as the combined viewpoints of the existing stakeholders. Merging process is repeated similarly to the above described incremental merging process between models from the new stakeholders and models from the merged ranked structure, until a successful model is found.

If there is no agreement reached when it comes to the highest rank of the merged ranked structure, then all the stakeholders, including the existing ones and the new ones, shall give their models at the next preference rank higher than the rank where the existing stakeholders reached the agreement. The same process is continued until a satisfactory model is found. The algorithm for procedure IteratedMerge() below represents this process.

---

procedure IteratedMerge()

---

inputs:

1. A set of partial ranked structures,

   $\{\text{NRS}_j \mid j \in \text{NEWSTAKEHOLDERS}\}$

2. A single partial ranked structure PM output from procedure

   IncrementalMerge()

3. A merging operator OP and associated function $\textbf{PartialMerge}_{OP}$

4. A set of CTL properties $PROP$

outputs:

1. A single partial ranked structure NPM, represented as a sequence of sets

   $\langle\ S_0,\ S_1,\ \ldots,\ S_r\ \rangle$, organized in ascending order of rank, where each

   set S_n contain models at rank n.

2. A model m

$done$:= false

for n = $0, \ldots, r$ **do**

for each new stakeholder $j \in$ NEWSTAKEHOLDERS

elicit all models at rank n and place them in the set NSM_n^j

$TEMP\_PM := \{PM_l \mid l \in \{0,\ \ldots,\ r\}\}$

$NPM :=$ $\textbf{PartialMerge}_{OP}(\{NSM_n^j \mid j \in \text{NEWSTAKEHOLDER},$

$n \in \{0,\ \ldots,\ r\} \bigcup TEMP\_PM\})$ where $NPM = \{S_0, \ldots, S_n\}$

if the S_{n-1} exists and is non-empty

if there is a model m in S_{n-1} that satisfies all properties in $PROP$

$done$:= true

return m, NPM

else

if there exists a model $m^{'} \in S_n$ that satisfies all properties in *PROP*

done := true

return $m^{'}$, *NPM*

if NOT *done*

STAKEHOLDERS := STAKEHOLDERS $\bigcup$ NEWSTAKEHOLDERS

call procedure **IncreamentalMerge**()

This last call to the procedure IncrementalMerge can be optimized (and is optimized in the implementation) by not starting the incremental elicitation process from rank 0, but from one rank higher than the highest rank r elicited (from both the prior set of stakeholders and the new set of stakeholders). Note that when the IncrementalMerge procedure is called, every stakeholder in the set STAKEHOLDERS $\bigcup$ NEWSTAKEHOLDERS will have completely specified all ranked structures up to rank r.

Thus, the algorithm embodies our framework's characteristics, which are interleaved elicitation, interleaved merging and interleaved model checking. It demonstrates the applicability of merging operation to inconsistency management in requirements engineering.

3.5 Example

For better understanding, we present an example to demonstrate how our framework works. In this example, we assume that arbitration operator Δ_{min} is chosen. We start by taking the sample viewpoints shown in Figure 3.1. Consider a set of ranked structures $V = \{\Phi_1, \Phi_2\}$, where Φ represents the viewpoints of stakeholder 1 and

Φ_2 represents the viewpoints of stakeholder 2. Stakeholder 1 first gives her most preferred model u_1 and stakeholder 2 gives her most preferred model u_2. So in the partial ranked structure Φ_1 of stakeholder 1 there is one model u_1 assigned rank 0 and there is one model u_2 assigned rank 0 in the partial ranked structure Φ_2 of stakeholder 2. Then, these two viewpoint models are checked for agreement, i.e. whether they are identical or consistent, based on the consistent rules described in section 3.4.

Before consistency checking, we need to define a signature map to unify the vocabularies of these two models. We take the same signature map from [11] as shown in Table 3.1. Obviously, these two models are inconsistent according to the above consistency rules as there are conflicts between the state transitions. As this is the case, both stakeholder 1 and stakeholder 2 must now give their models of next preference level, i.e. level 1. The models they give are u_3 and u_4 respectively. Models of Φ_1 (u_1, u_3) and models of Φ_2 (u_2, u_4) are checked again for agreement. Assume no agreement is reached, then both stakeholders have to give their models of preference level 2, which are u_5 and u_6 respectively. Now Φ_1 ranks the models as $\Phi_1(u_1) = 0$, $\Phi_1(u_3) = 1$, $\Phi_1(u_5) = 2$ and Φ_2 ranks the models as $\Phi_2(u_2) = 0$, $\Phi_2(u_4) = 1$, $\Phi_1(u_6) = 2$ (see Figure 3.6 (a) and (b) for a pictorial view of the ranked structures of Φ_1 and Φ_2 respectively). Suppose an agreement is reached at this preference rank (rank 2), assuming u_3 and u_6 are consistent, u_5 and u_2 are consistent, and u_5 and u_6 are consistent, even though they are not identical. Then merging operation is performed to combine these three sets of consistent models and assign ranks to the combined models, hence creating a new ranked structure, which we refer as Φ. We name the model combined from u_3 and u_6 as v_1, model combined from u_5 and u_2 as v_2 and model combined from u_5 and u_6 as v_3. With the merging operator Δ_{min}, model

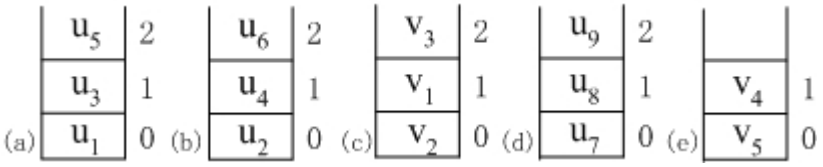

Figure 3.6: Partial Ranked Structures

v_1 is assigned the rank 1, model v_2 assigned rank 0 and model v_3 assigned rank 2. Therefore, the ranked structure Φ contains models v_2, v_1 and v_3 at ranks 0, 1 and 2 respectively (see Figure 3.6 (c) for a pictorial representation).

As v_2 is the most preferred combined model, it is first checked against a set of system properties using SMV model checker. If v_2 satisfies all the system properties, it will become the desired resulting model and the final outcomes are v_2 and ranked structure Φ. Otherwise, we have to go to the next most preferred model and so on, until we find a successful one. If none of the models in Φ satisfies the system properties, then the stakeholders have to continue to provide their models of the next preference rank, which is rank 3. The process is repeated until a satisfactory model is found. In our example, we assume combined model v_1 between models u_5 and u_2 is the successful model, therefore the final outputs are v_1 and ranked structure Φ.

After the completion of the process, if there is another stakeholder 3 coming and presenting her viewpoint model u_7, then the process should continue by checking model u_7 with model v_2, the most preferred one in the ranked structure Φ for consistency. If an agreement is reached, then model v_1 and model u_7 are combined into a single model, which will be taken for model checking. If it returns a true value, such a model is the output model with a ranked structure containing only this model. Assuming in this example, model u_7 is not consistent with v_2, then stakeholder 3 has

to give her next most preferred model which is u_8 at rank 1. Models at rank 1 of the ranked structure Φ shall also be taken to join the consistency checking process. That is models of $\Phi(v_2,\ v_1)$ and $\Phi_3(u_7,\ u_8)$ are checked for consistency. Assuming an agreement is reached at this rank, i.e. model v_1 and model u_8 are consistent, then they are combined into a single model, which is named as v_4. A new ranked structure $\Phi^{'}$ is formed containing model v_4 at rank 0. Now model v_4 is model checked against the system properties. Assuming that it does not satisfy all the properties, then stakeholder 3 has to give her model (u_9) of next preference rank, i.e. rank 2. Model of rank 2 of ranked structure Φ is also taken for consistency checking as we did for the previous ranks. Assuming model u_9 is consistent with model v_2, then they are combined into model named v_5 and assigned rank 0 with merging operator Δ_{min}. Contents of ranked structure $\Phi^{'}$ is changed and it contains models v_5 at rank 0 and v_4 at rank 1 (see Figure 3.6 (e)). If model v_5 satisfies all the system properties, the final outcomes are model v_5 and ranked structure $\Phi^{'}$. Otherwise, the merging process need to continue with all the stakeholders, i.e. stakeholders 1, 2 and 3 providing their models at next preferred rank, rank 3. Now the $V = \{\Phi_1,\ \Phi_2,\ \Phi_3\}$ and the process is iterated as described above until a satisfactory combined model is found.

Chapter 4

Implementation

In this chapter, we present a viewpoint merging tool based on the framework presented in the previous chapter. First, we describe the design using an object-oriented approach. The design consists of three major components. Details of these components and their relationships are explained in section 4.1. Then we present the implementation of the algorithms presented in Chapter 3, operationalized through a user interface written in Java (a object-oriented programming language). Individual method is explained in detail in Section 4.2.

4.1 System Design

The overall architecture of the merging system consists of three components: Model Elicitation, Model Merging and Model Checking.

The Model Elicitation system relates to model elicitation from stakeholders. In order to maximize convenience to stakeholders, the stakeholders are able to present their models at their own terminals to carry out the followings.

- Register as a new stakeholder providing their required details as well as their

viewpoint-specific vocabularies.

- Registered stakeholders are able to login to the system where they will be able to view their models.
- Registered stakeholders are able to enter details of the models as well as their associated preference ranks (stakeholders do not enter ranks of the models they specify - instead the system prompts them for models of a given rank).
- Registered stakeholders are able to edit the models entered except for the model ID.
- Registered stakeholders are able to delete a model.
- Registered stakeholders are able to modify their personal details including the password.

After all stakeholders have registered (at which time they provide their viewpoint-specific vocabularies), the analyst is required to create a *signature map* (described in Chapter 3).

The second component of the system is to merge the models elicited from the stakeholders and produce a ranked structure. It carries out the following functions.

- Signature mapping. Models from different stakeholders are mapped to the same vocabulary according to the signature map created by the system analyst.
- Most of the functionality described in the **IncrementalMerge**() procedure described in Chapter 3 is implemented in this component, but involves, in part,

the model elicitation functionality of the first component and the model checking functionality of the 3rd component.

- Consistency checking, which implements the **Consistent**() procedure described in Chapter 3.

- Model combination. Combine a set of consistent models into a single model and assign a rank to this model (based on the merging operator being used and the ranks of the input models). It implements the **Combine**() function described in Chapter 3.

The last component is model checking which perform the function to check models against a set of system properties. In this system, the most preferred model from the newly created ranked structure will be taken for model checking.

In the following, we will discuss the data structure to implement the merging system consisting of the above three components.

4.1.1 Data Structure Description

Figure 4.1 shows the class diagram describing the above three components.

The viewpoints used in our framework are represented as finite state models, each of which consists of a set of states and transitions. The class *Model* is used to describe the finite state models. It has aggregated classes *Node* and *Arc* that describe states and transitions respectively. Class *Node* has aggregated class *Variable* that represents the atomic propositions that are assigned a value of true or false in each state. Therefore, all these classes record details of a complete model. The models

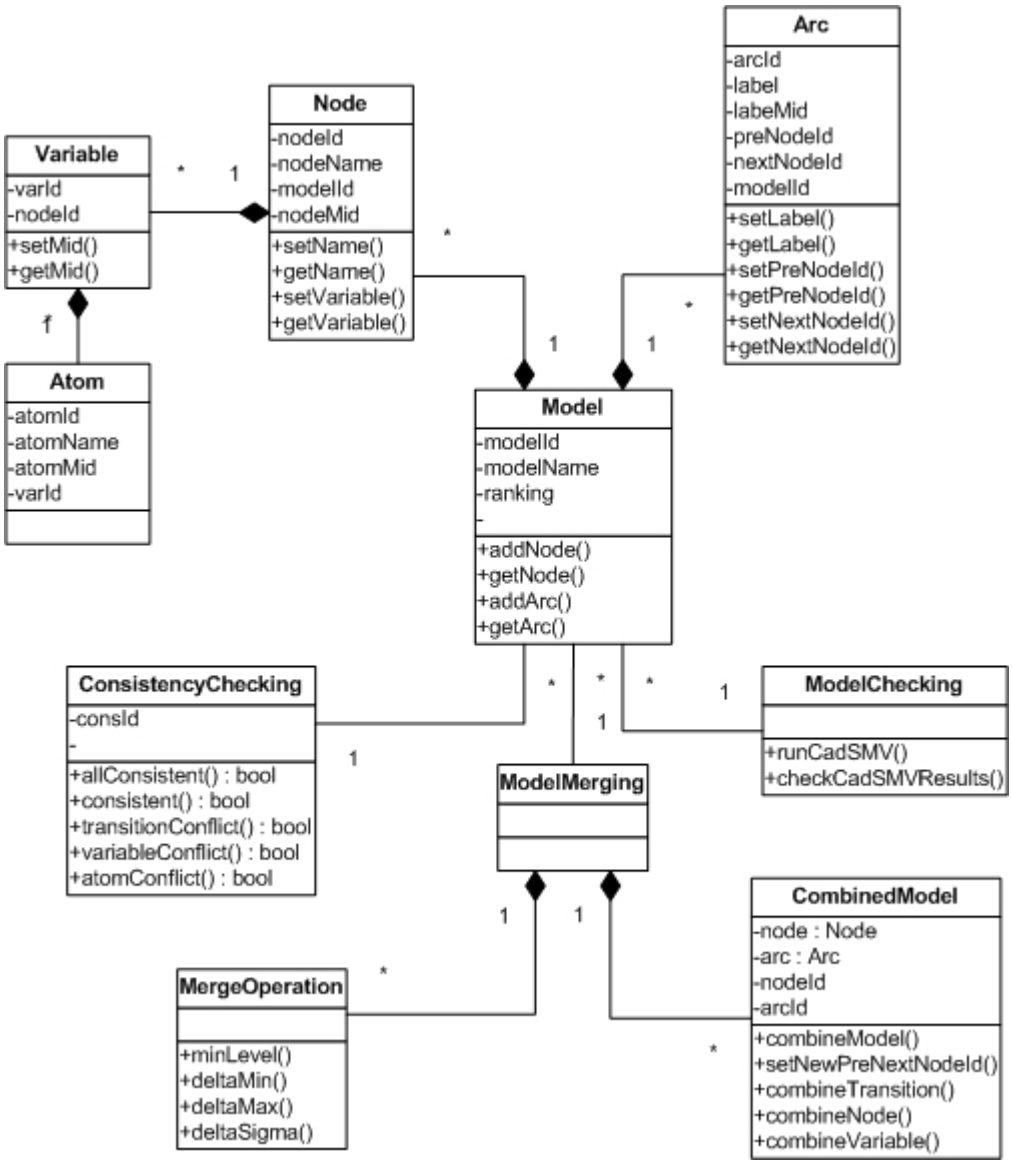

Figure 4.1: Class Diagram of the proposed Framework

are stored in a database. The class *Model* implements the functions related to Model Elicitation component.

The class *ModelMerging*(of Model Merging component) interacts with the class *Model*. Its aggregated class *Consistent*, which implements the consistency checking function of the Model Merging component, checks models retrieved from the database for consistency. Models of one stakeholder are checked with those of another stakeholder. In Java, we use the "vector" data structure to store the models. Every element of one vector holds models of one stakeholder. The other aggregated class *Combine* combines a consistent set of models into a single one. Model of one stakeholder is combined with model of next stakeholder and the merged model is combined with model of next stakeholder and so on. Rank of the combined models are generated based on the merging operator used. The final combined model is stored in the database with new name and ID. Its states, state variables and transitions are also recorded to the database. This class implements the model combination function of the Model Merging component. Therefore, the class *ModelMerging* implements the functionality to produce a partial ranked structure containing the combined models with their associated preference ranks.

The class *ModelChecking* has a function to call SMV model checker to check the combined models and a function to check the output returned by SMV model checker to decide whether the combined model is satisfactory or not.

4.2 Implementation Description

4.2.1 Overview

The prototype is written in Java. We have designed a user interface for data entry which is used by the stakeholders to enter their requirements/preferences (i.e. their ranked structure). The data input are stored in a database. Different stakeholders can enter their models at their individual terminals. Once all the stakeholders have given their models, the system is able to automatically check the models and perform the merging operation if a consistent set of models is found. In our prototype, Microsoft (MS) Access is used to create the database, which consists of twelve tables. Schema definition in SQL of these tables are listed below:

- **create table** *stakeholder* (*SHId* **char(5) not null**, *SHName* **char(30)**, *SHAddress* **char(30)**, *password* **char(6)**, **primary key** (*SHId*))
- **create table** *model_sh* (*SHId* **char(5) not null**, *modelId* **char(5) not null**, **primary key** (*SHId*, *modelId*))
- **create table** *model* (*modelId* **char(5) not null**, *modelName* **char(30)**, *consId* **char(5)**, *rank* **integer**, *operator* **char(20)**, *desc* **char(30)**, **primary key** (*modelId*))
- **create table** *node* (*nodeId* **char(5) not null**, *modelId* char(5) not null, **primary key** (*nodeId*, *modelId*))
- **create table** *variable* (*varId* **char(5) not null**, *value* **boolean**, *nodeId* **char(5) not null**, *modelId* **char(5) not null**, **primarykey** (*varId*, *nodeId*, *modelId*))

- **create table** *arc* (*preNodeId* **char(5) not null**, *nextNodeId* **char(5) not null**, *modelId* **char(5) not null**, **primary key** (*preNodeId*, *nextNodeId*, *modelId*))
- **create table** *consModel* (*consId* **char(5) not null**, *modelId* **char(5) not null**, **primary key** (*consId*, *modelId*))
- **create table** *nodeVocabulary* (*SHId* **char(5)**, *name* **char(30)**, *nodeId* **char(5) not null**, **primary key** (*nodeId*))
- **create table** *varVocabulary* (*SHId* **char(5)**, *name* **char(30)**, *varId* **char(5) not null**, **primary key** (*varId*))
- **create table** *nodeSignatureMap* (*nodeId* **char(5) not null**, *mapId* **char(5)**, **primary key** (*nodeId*))
- **create table** *varSignatureMap* (*varId* **char(5) not null**, *mapId* **char(5)**, **primary key** (*varId*))
- **create table** *loginSH* (*SHId* **char(5)**, *rank* **integer**, *newSH* **boolean**)

In the following discussion, we will refer to the notions of *source model* and *combined model*. A *source model* is a model that was originally specified by a stakeholder while a *combined model* is one that is output by the **Combine**() function described in Chapter 3.

Table STAKEHOLDER records details such as id, name, address and login password of all the registered stakeholders. Table MODEL_SH joins tables STAKEHOLDER and MODEL together. It only contains two fields: stakeholder ID and

model ID. Table MODEL records both source models and combined models. It contains model ID, model name, *consId* of consistent set of models (we identify maximal consistent sets of models from the input set of models and each such set is identified by *consId*). If the model is a combined model, it records from which set of consistent models it is combined in the *desc* field. If it is a source model, its *consId* is null. Table MODEL also records the rank of each model and the merging operator used to obtain the combined model (if the model in question happens to be a source model - the value is null). Table NODE records the node ID and ID of its associated model. Similarly, table VARIABLE records the information of a variable: variable ID and ID of its associated node and the true or false value that variable holds in the node. Table ARC records the arc ID, arc's pre node's ID and next node's ID as well as its associated model ID. Details of the consistent models are recorded in table CONSMODEL, which contains the ID for the set of consistent models and the ID of models contained in that set.

Table NODEVOCABULARY captures node-specific vocabularies provided by the stakeholders. It contains stakeholder ID, the node names specified by the stakeholders and node Id that distinguishes the names given by the stakeholders. Table VARVOCABULARY records the variable-specific vocabularies given by the stakeholders and it contains stakeholder ID, variable name and variable ID. From the vocabularies recorded in these two tables, the analyst creates the respective tables NODESIGNATUREMAP and VARSIGNATUREMAP to unify the vocabularies of nodes and variables given by the stakeholders. Each table contains the respective ID and the mapping value.

The last table LOGINSH registers the stakeholders that already provide their models of a particular rank and also record whether the stakeholders are newly joining one or the existing ones.

In the subsection that follows, we will give a detailed description of the user interface that facilitates our merging system.

4.2.2 Implementation Description

The data input would ideally be a diagrammatic state machine editor. Implementing such an editor is a non-trivial task, and the exercise was deemed to be orthogonal to the objectives of this research. We have therefore devised a simple form-based interface for specifying state machine models.

Data input interface has five forms, i.e. 1) stakeholder login form, 2) stakeholder registration form, 3) data entry main menu, 4) add model form, 5) add state form and 6) add arc form. We will explain the form with an example. In this example, there are two stakeholders named Alice and Bob.

The initial interface presents the user with two options, one for a new stakeholder registration, the other for existing stakeholders. Selecting the "New User" option will take the user to the Stakeholder Registration Form.

The Stakeholder Registration System captures information of a stakeholder as listed on Figure 4.2. It also captures viewpoint-specific vocabularies presented by the stakeholders. Stakeholder details are recorded to the STAKEHOLDER table in the database. The stakeholder ID is generated automatically based on the value of the last

Figure 4.2: Stakeholder Registration Form

id in the database incremented by 1. The vocabularies are added in the corresponding vocabulary tables, i.e. node vocabularies to table NODEVOCABULARY and variable vocabularies to table VARVOCABULARY. We assume that the analyst also registers as a stakeholder, and provides a vocabulary. This is the vocabulary used to represent combined models.

For the registered stakeholders, after logging in based on the user's ID and password, the user is presented with three options , where they can choose whether to

Figure 4.3: Add Model Form

add model, edit model or delete model. In our prototype[1], we have only implemented the "Add Model" and "Delete Model" functions.

The merging process begins with all stakeholders being prompted to provide all of their most preferred models, i.e. models at rank 0. If a consistent outcome is found from these models, that also satisfies a given set of properties specified in Computation Tree Logic (CTL), then the process terminates. Otherwise, all stakeholders are prompted for their next most preferred models, i.e. models at rank 1. The process follows the steps described in the **IncrementalMerged**() procedure in Chapter 3.

When the user selects the "Add Model" option, the Add Model form is displayed as shown in figure 4.3.

The stakeholder ID and name are retrieved from the database and shown automatically. The Model ID is generated automatically as the stakeholder ID is. The

[1]See Appendix A for the complete java source codes

preference rank is prompted by the system and the user only need to enter the name of the model. In this example, the preference rank is 0 and model name is u1 for stakeholder Alice. The partial ranked structure of stakeholder Alice is therefore elicited.

Models are added in two stages. In the first stage, state details are added; in the second stage, transition/arc details are added. Selecting the "Add State" option will call the Add State form (see Figure 4.4) for the user to select her specified state names and variable names from their respective drop down lists. Each variable can take the value of *true* or *false*. Therefore, there is also a drop down list containing "True" and "False" that can be selected to assign the value to the variables. After the user selects the a state name, a variable name and a value and select the "Add" option, the variable with its assigned value is displayed in the text area below. The next variable is then entered if there is any. A particular variable can also be removed. Just select that variable listed in the text area and select the "Remove" option.

After finishing entering all the variables for that particular state, the user selects the "Save State" option to save the data for this state. Then the user enters the next state following the same procedure. In our example, Model $u1$ has two states: $A0$ and $A1$, and two variables: x and a. Both x and a hold in state $A0$(See Figure 4.5(a)). On entering this information, it is saved and state $A1$ containing variables x and a both with false value is added. Figure 4.4 shows the input of state $A1$. The user then returns to the Add Model form and selects the "Add Arc" option to enter the transition details.

The state names and variable names selected are mapped automatically according to the signature map already created by the analyst. In this example, we follow the

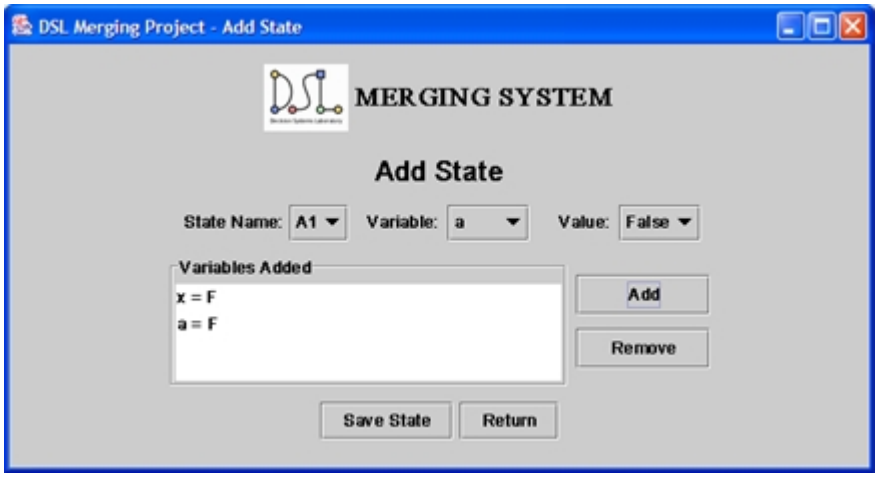

Figure 4.4: Add State Form

signature map listed in Table 4.1

The transition details are added in a similar way as the user adds the state details. The user selects the names of the arc's in-node and out-node from the respective dropdown lists (see figure 4.6). Names of these nodes come from the nodes added via Add State form. After all the transitions have been entered and saved, the user returns to the Add Model form and select the "Save Model" option inserting the complete model into the database. Model, node and arc data are inserted to the MODEL table, NODE table, VARIABLE table and ARC table respectively. Then a message dialog box is displayed to ask whether the user wants to add another model of the same preference rank. The user can enter another model of the same preference rank if there is one. As stakeholder Alice has no more model at rank 0 to add, once she selects the "No" option, she returns to the main menu.

Selecting the "Delete Model" option will take the user to the Delete Model Form where models of that particular user are displayed. Displayed details include the

Table 4.1: Signature Map

S_1	S_2	Mapping
A_0	B_0	N_0
A_1	B_2	N_1
-	B_1	N_2
x	z	a
a	y	b

model ID, the model name and the associated preference rank. The user can select the model to be deleted. In this system, the "Add Model" option is the major concern and we will continue its discussion in the following.

In our example, it is now stakeholder Bob's turn to given his model at rank 0. The model given by Bob is named $u2$ as shown in Figure 4.5(b). Once he finishes entering his model, he is also prompted a message dialog box to select whether to add another model. When he selects "No" option, consistency checking is performed as the system can detect that he is the last stakeholder to enter the models (i.e. all the stakeholders have entered their models at rank 0). Consistency checking is performed based on the consistency rules described in the previous chapter. If no agreement is reached, the message is displayed using a dialog box on the screen and all the stakeholders involved are informed, in which case, the stakeholders have to provide their models of next preference rank. If there is an agreement, merging operation will continue by combining any consistent sets of models according to the algorithm of **Combine**() function described in Chapter 3.

In our example, these two models are not consistent, therefore, stakeholders Alice and Bob have to present their models of next preference rank, i.e rank 1. Once

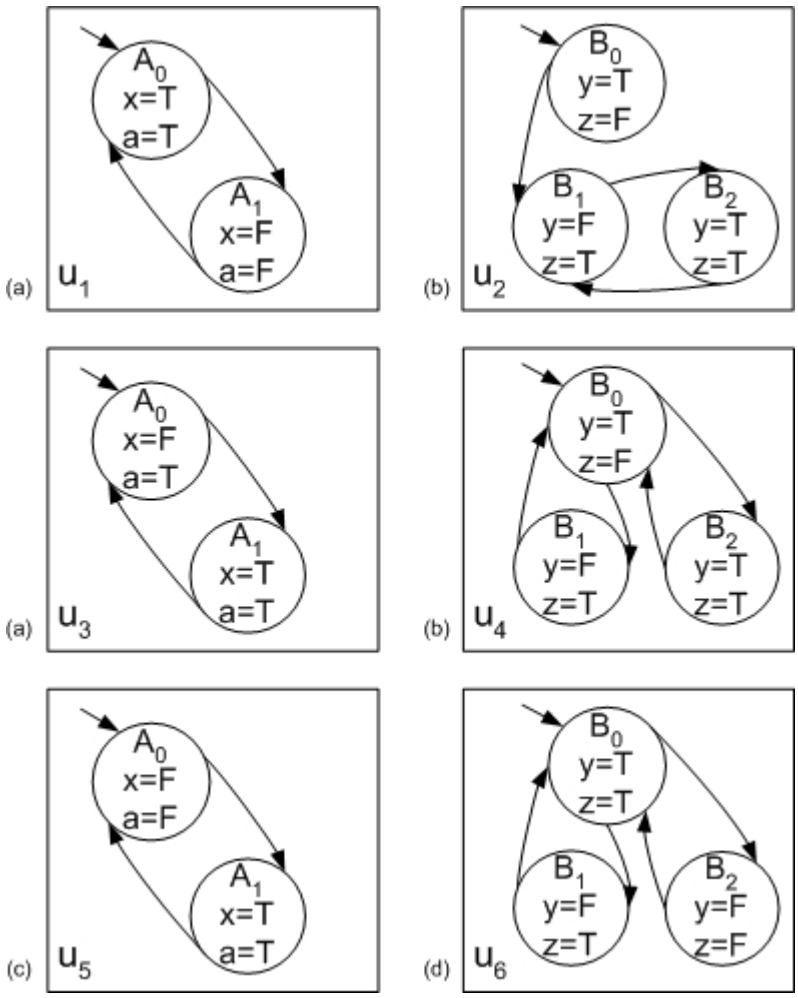

Figure 4.5: Example Viewpoints

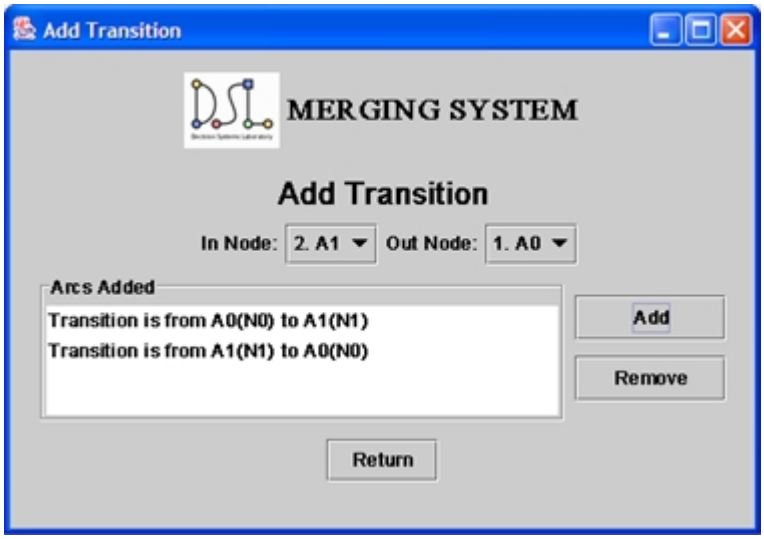

Figure 4.6: Add Transition Form

they have provided their models $u3$ and $u4$ (as shown in Figure 4.5 (c) and (d)) respectively, consistency checking is reapplied. This time consistent set of models is found, i.e. model 3 and mode 4 are consistent. Details of the consistent set of models are recorded to the CONSMODEL table in the database. Merging operation continues. We assume Δ_{min} merging operator is used. Therefore, the above merging process implemented the algorithm $\mathbf{PartialMerge}_{\Delta_{min}}()$.

Result of the merge is displayed in a separate form, which will show the contents of the newly created ranked structure. It displays the preference ranks, model IDs of the merged models, merging operator used and a description indicating the model IDs of the source models that form the combined models. In this example, only one combined model is created, which is assigned the rank 0, model ID 1001 (combined models are assigned the model IDs greater than 1000 to distinguish from the source models), with "Delta min" and a description of "models 3 and 4" recorded to the

MODEL table (see Figure 4.7). Its nodes, variables and arcs are inserted respectively into the tables NODE, VARIABLE, and ARC in the database.

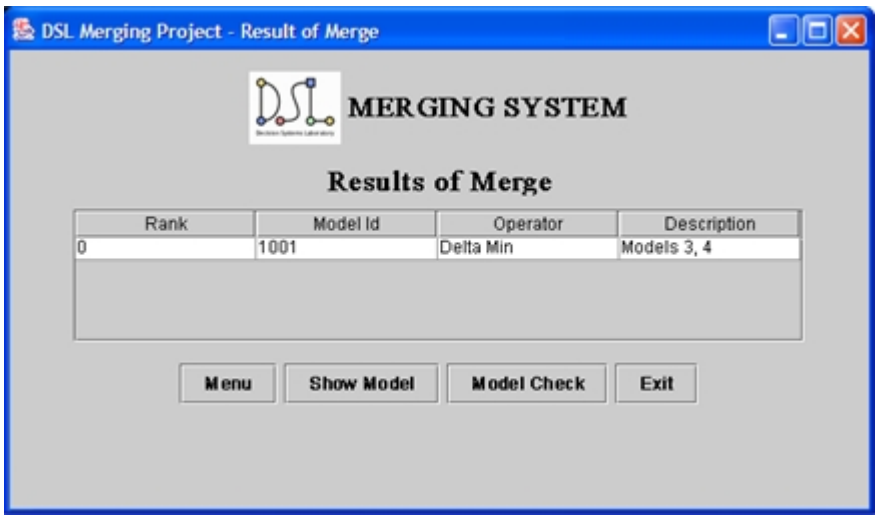

Figure 4.7: Model Merging Results Form

The combined model is then selected for model checking against the system properties. The SMV model checker is used for checking system properties (these are assumed to be written in CTL). The SMV system assumes that models are specified in a system-specified input language - called the SMV language. We use an intuitive representation of finite state models in a format that users can easily understand and that supposes a diagrammatic representation of such models (although we have not implemented the graphical display component). Translating between this representation and the SMV language is non-trivial, and implementing such a translator is not central to our research aims. Consequently, we have not implemented such a translator, but we point out that such a translator could be implemented. For our prototype, we assume that the analyst performs this translation. The model checking

component thus functions as a stand-alone system in the current version of our prototype. In this example, we assume the satisfactory model should satisfy the following property: For any state, if a a holds, then it will eventually get to a state where b is false. The property can be represented in CTL (see chapter 3 for more details) as: AG(a $\rightarrow$ AF ($\sim$b)).

Figure 4.8 (a) is the graphic view of the combined model between models 3 and 4. We name this model as v_1 and it is represented in the following SMV code.

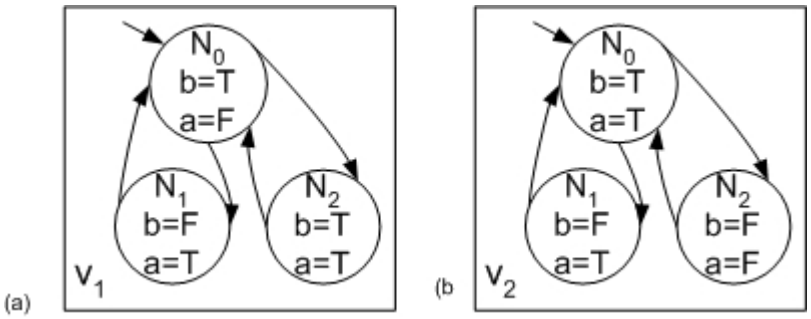

Figure 4.8: Combined Models

MODULE main

VAR

a, b: boolean;

ASSIGN

init(a) := 0;

init(b) := 1;

next(a) := $\sim$a;

next(b) := {0, 1};

SPEC

AG (a → AF (∼b))

When the SMV model checker is run to check model v_1, it returns a false value, indicating that the model does not satisfy the above property (See Figure 4.9 for part of the verifying result). Therefore, stakeholders Alice and Bob are informed of the result and they must give their models again of next preferred rank, i.e. rank 2. The models given by Alice and Bob in our example are $u5$ and $u6$ respectively as shown in Figure 4.5(e) and (f). Once the models are input following the same procedure described above by Alice and Bob, consistency checking is performed. Consistent set of models is found between model 1 and 6, in addition to the consistent set of model 3 and 4. See Figure 4.10 for the merge results.

A new ranked structure is now created containing two combined models. Model 1001 formed between model 1 and 6 (see Figure 4.8(b) for a graphic view) is assigned the rank 0 and model 1002 formed between model 3 and 4 (see Figure 4.8(a)) is assigned the rank 1. We name model 1001 as v_2 and model 1002 remains as v_1.

Obviously, model v_2 ranked 0 is sent to SMV model checker to be checked against the property. The following is the SMV's representation of this model.

```
MODULE main
VAR
    a, b: boolean;
ASSIGN
    init(a) := 1;
    init(b) := 1;
    next(a) := {0, 1};
```

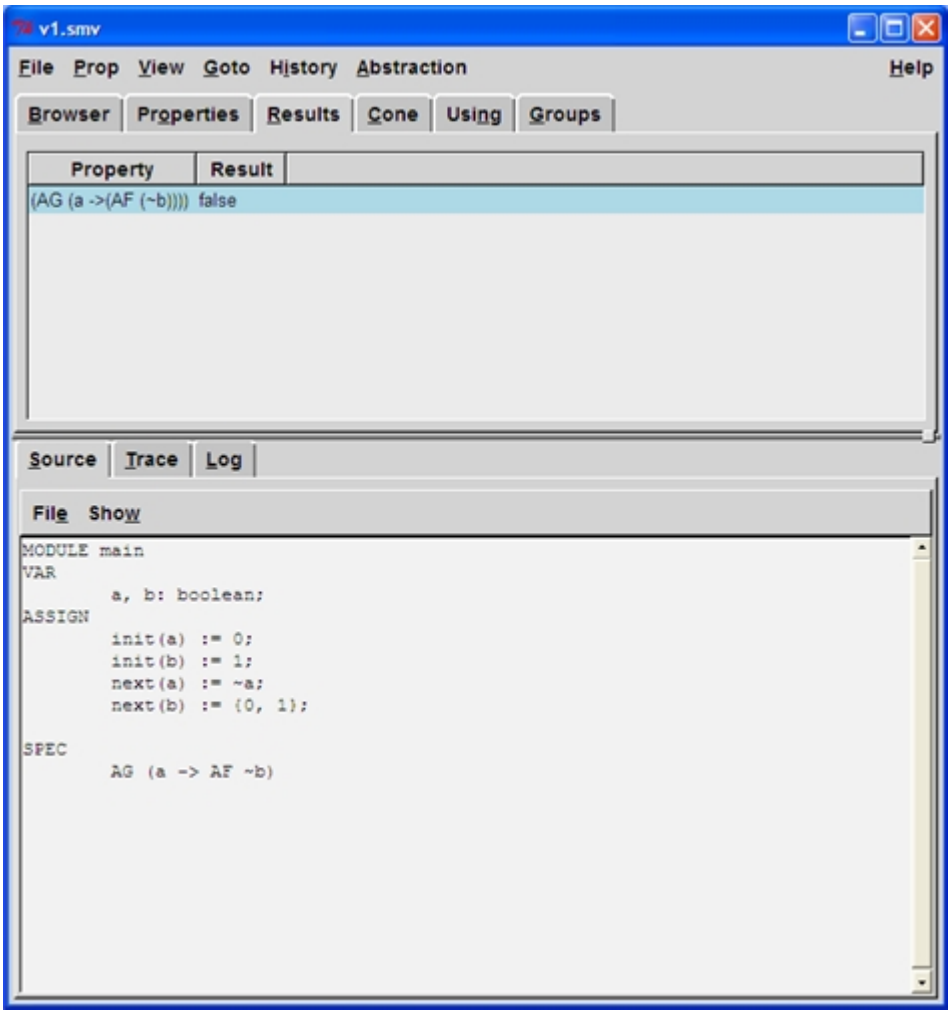

Figure 4.9: SMV Running Output for Model v_1

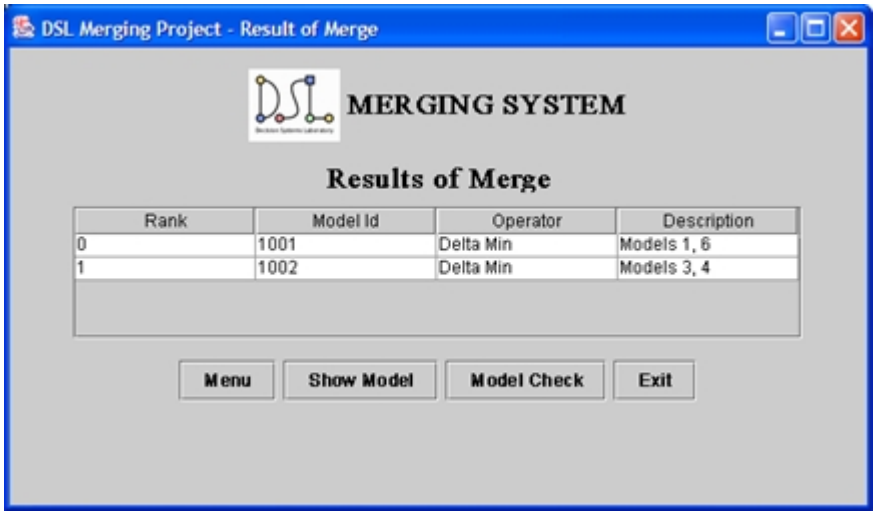

Figure 4.10: Second Merging Results Form

next(b) := ∼y;

SPEC

AG (a → AF (∼b))

After running SMV model checker, a true value is returned (see Figure 4.11) indicating the model satisfies the property. Therefore, the merged model v_2 between model 1 and 6 becomes the resulting model. No further iteration is required. The merging system implementing the algorithm **IncrementalMerge**() described in Chapter 3 output model v_2 and a ranked structure containing model v_2 with rank 0 and model v_1 with rank 1.

In the cases when new stakeholders are joining, they enter their models in the same procedure as described above and their models are then checked with the most preferred models in the ranked structure, i.e. model v_2 of model ID 1001. Analysis and model merging procedure are repeated until a satisfactory combined model is found, which is implemented based on the algorithm **IteratedMerge**() described in

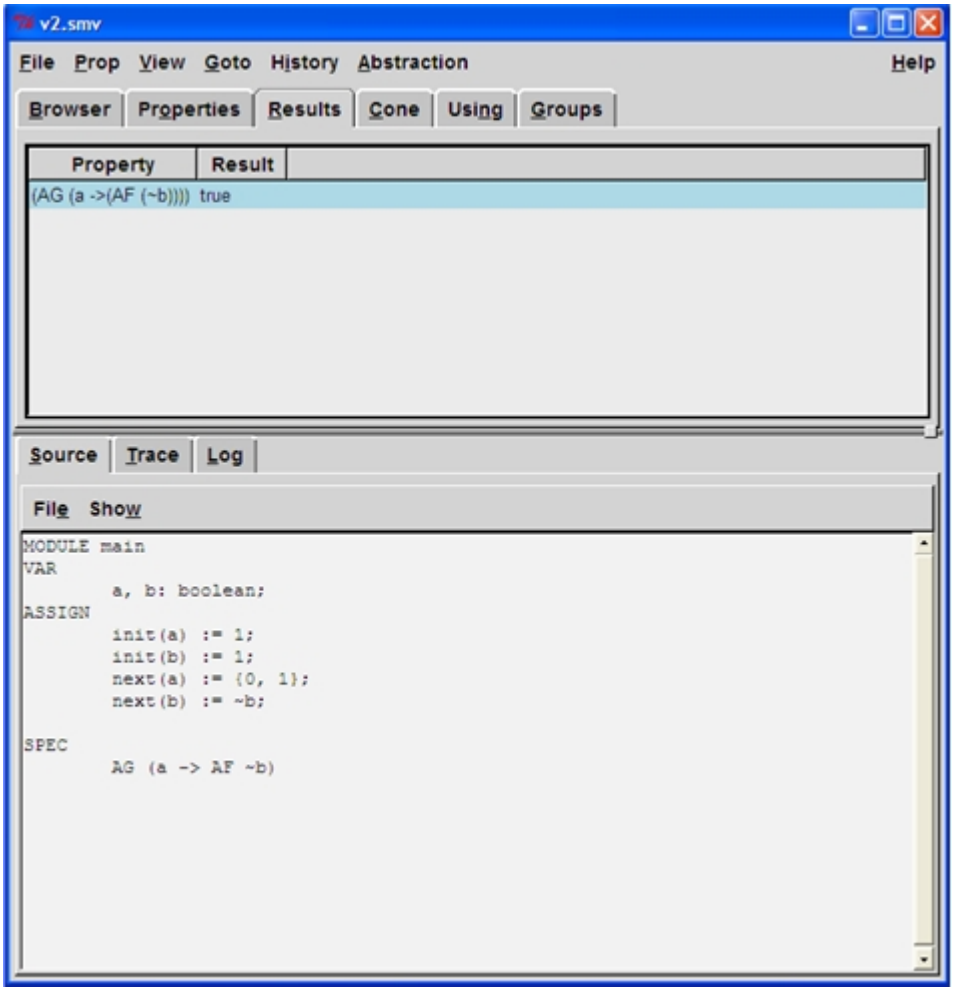

Figure 4.11: SMV Running Output for Model v_2

Chapter 3.

Chapter 5

The Case Study

In this chapter, we present two case studies involving the application of our viewpoint merging approach.

5.1 Telephone System Case Study

The purpose of this case study is to further demonstrate the applicability of belief merging to inconsistency management in requirements engineering. This case study was the result of a real life experience involving two stakeholders who provided their viewpoints on the behavioral specification of a telephone system.

The case study involved two individuals (we shall refer to them as Tom and Jerry in the following) who were given an initial problem description as plain text English. They used the State View Merge System to merge their initially inconsistent viewpoint (based on their diverging interpretation of the textual description given to them).

5.1.1 The Scenario

The scenario involves the telephone handset being used to receive a call. The following was the description of the requirement given to the stakeholders.

A telephone handset can be used to make and receive a call. When it is idle, the receiver is replaced. When there is an incoming call, it is connected. If the incoming call is not answered, it will be disconnected and become idle again.

Based on the above description, two stakeholders first presented their most preferred viewpoints, i.e. viewpoints at rank 0. It is assumed that their requirements could reach an agreement at some rank and then are combined to give a complete description of the handset.

Consistency rules described in Chapter 3 are used to test whether the different state transition diagrams representing the same device are consistent with each other.

The merging process described in Chapter 3 combines the set of consistent models into a single one and then a set of system properties applied to this telephone system will be used to check whether the resultant combined viewpoint is a satisfied and desired viewpoint.

The complete experiment was conducted using our tool developed based on the algorithms described in Chapter 3.

5.1.2 Experiment Description

The case study was developed based on the viewpoints on [11] between Tom and his colleague Jerry. Tom's most preferred viewpoint suggests replacing the receiver during

Table 5.1: vocabulary Map

Tom	*Jerry*	Mapping
DialTone	DIALTONE	DIALTONE
Idle	IDLE	IDLE
Ringing	RINGING	RINGING
Connected	CONNECTED	CONNECTED
offhook	offHook	offhook
connected	Connected	connected

an incoming call will not disconnect the call (see model u_1 in Figure 5.1(1)). From the Figure 5.1 (1) we can see there was a transition from state **Connected** to state **Ringing**, where variable *Connected* was true in both states. Jerry's most preferred viewpoint assumed that replacing the receiver always disconnects the call (see model u_2 in Figure 5.1(2)), where there were transitions from states **CONNECTED** and **DIALTONE** in which variable *offHook* held to state **IDLE** in which variable *offHook* was false and *Connected* was false. Jerry also had another viewpoint model u_3 at rank 0 as shown on Figure 5.1.(3). In this model, he had a transition from state IDLE to state DIALTONE, indicating that it was allowed to make a call.

Tom's diagram had four states, **DialTone**, **Idle**, **Connected** and **Ringing**, and two variables: *offhook* and *connected*. Jerry's diagram was also represented in four states: **DIALTONE**, **IDEL**, **CONNECTED** and **RINGING**, with two variables: *offHook* and *Connected*. Their vocabularies were mapped according the signature map on Table 5.1. Tom's ranked structure Φ_1 contained a model u_1 rank 0 and Jerry's ranked structure Φ_2 contained two models u_2 and u_3 at rank 0. Once their viewpoints were presented, i.e. they both entered their requirements that were recorded to the database, consistency checking was performed immediately and a message dialog box

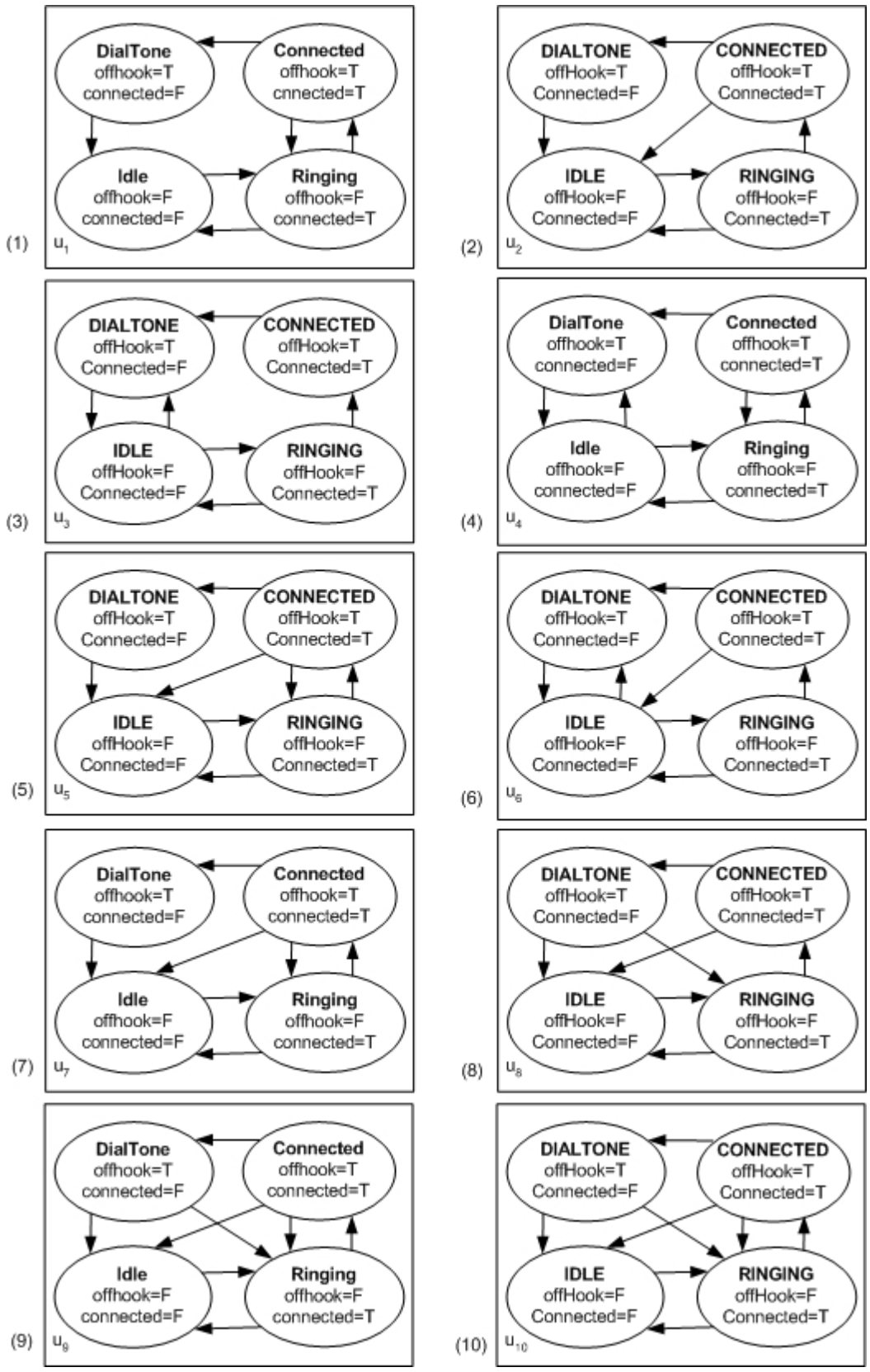

Figure 5.1: All Viewpoints Elicited for Telephone Systems

popped up to inform the users that no consistent model was found, indicating model u_1 was not consistent with model u_2 or model u_3.

Then the two stakeholders had to present their next preferred viewpoints. When the "Add Model Form" was invoked, the system indicated that viewpoints at rank 1 were required from the stakeholders. Tom entered his viewpoint u_4 at rank 1 as shown on Figure 5.1(4). He changed his viewpoint by adding a transition from state **Idle** to state **DialTone**, which allowed making a call. Jerry also had two viewpoints at rank 1. The first one was u_5 (see Figure 5.1(5)). It allowed that replacing the receiver would either disconnect the call or still have the call connected by having transitions from state **CONNECTED** to states **RINGING** and **IDLE**. But Jerry's second viewpoint u_6 at rank 1 (see Figure 5.1(6)) retained his previous opinion that replacing the receiver would always disconnect the call, but he also allowed making a call as well. After these three models were entered, consistency checking was performed among models of $\Phi_1(u1,\ u4)$ and models of $\Phi_2(u2,\ u3,\ u5,\ u6)$. Still no consistent models were found.

Tom and Jerry kept entering their viewpoints of next preference rank, i.e. rank 2. Tom's viewpoint u_7 at rank 2 allowed that the call could be either disconnected or connected when the callee replaces the receiver, with transitions from state **Connected** to states **Idle** and **Ringing**. As it was the viewpoint of a callee, Tom believed that he should not be concerned with the feature of call making. Therefore, he removed the transition from state **Idle** to state **DialTone** (see Figure 5.1(7)). At this rank, Jerry presented only one viewpoint instead of two as he did in the previous ranks. The viewpoint he presented was u_8 (see Figure 5.1(8)). Jerry still preferred that replacing the receiver would disconnect the call, but he also allowed that when the user was

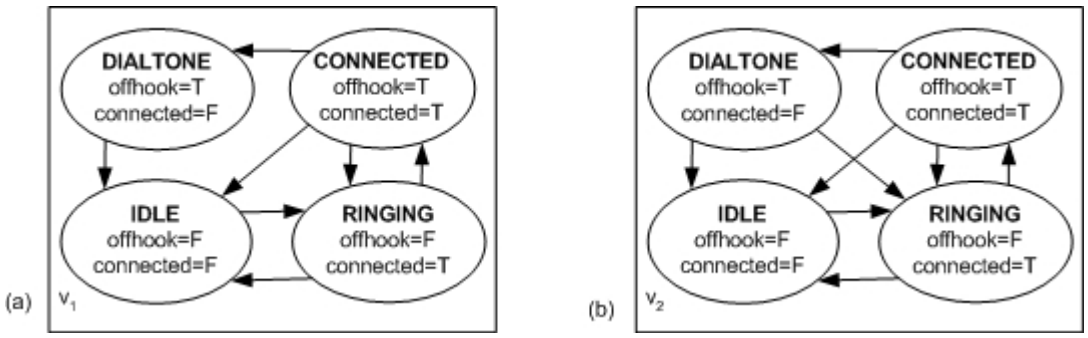

Figure 5.2: Result of Merge

dialing, if there was an incoming call, the user could stop making the call and receive the incoming call instead. At this rank, the tool detected that consistent models were found. Therefore a merging operation was performed (in this experiment, both Tom and Jerry agreed to use Δ_{max} merging operator) and the results were displayed in another window showing that a merged model ranked 0 was combined between model u_7 of Tom at his preference rank 2 and u_5 of Jerry at his preference rank 1.

As both Tom and Jerry had the same states and same variables, we can see that these two consistent models were actually identical. We rewrote the models as shown in Figure 5.2(a) with the signature map names.

Now the combined model was model checked against the properties using SMV. Both Tom and Jerry agree that the following properties should be satisfied.

Property 1 If you are connected, you can replace the receiver. It is represented in CTL as: AG(connected $\rightarrow$ EX($\sim$offhook)).

Property 2 If you are dialing, you can receive an incoming call. It is represented in CTL as: AG((offhook$\wedge\sim$connected)$\rightarrow$EX(connected)).

SMV model checker returned *true* for the first property, but *false* for the second property, indicating the model did not satisfy all the properties. Therefore, models of next preference rank needed to be elicited from the the stakeholders.

Therefore, Tom and Jerry had to enter their models of rank 3, which were models u_9 and u_{10} respectively (see Figure 5.1(9) and (10)). Now models of $\Phi_1(u_1,\ u_4,\ u_7,\ u9)$ and models of $\Phi_2(u_2,\ u_3,\ u_5,\ u_6,\ u_8,\ u_{10})$ (see Figure 5.3 (a) and (b) for a pictorial viewpoint of the ranked structures of Φ_1 and Φ_2 respectively) were checked for consistency. A window was displayed showing that two merged models were produced. One was ranked 0 (and formed by the combination of models u_7 and u_5), which was named v_1, the other was ranked 1 (and formed formed by the combination of models u_9 and u_{10}), which was named v_2. Therefore, a new merged ranked structure Φ was produced. See Figure 5.3(c) for the pictorial view.

As the merged model between models u_7 and u_5 did not satisfy all the properties, it would not be taken for model checking even it was ranked the most preferred model. So the next preferred model would be checked against the above mentioned properties using SMV model checker. It was model v_2 (formed by the combination of models u_9 and u_{10}). The SMV model checker returned *true* for both properties. Therefore, model v_2 was the output model.

Tom and Jerry finally resolved the problem at rank 3 with a model satisfies all the system properties and a merged ranked structure containing two models, model v_1 at rank 0 and model v_2 at rank 1.

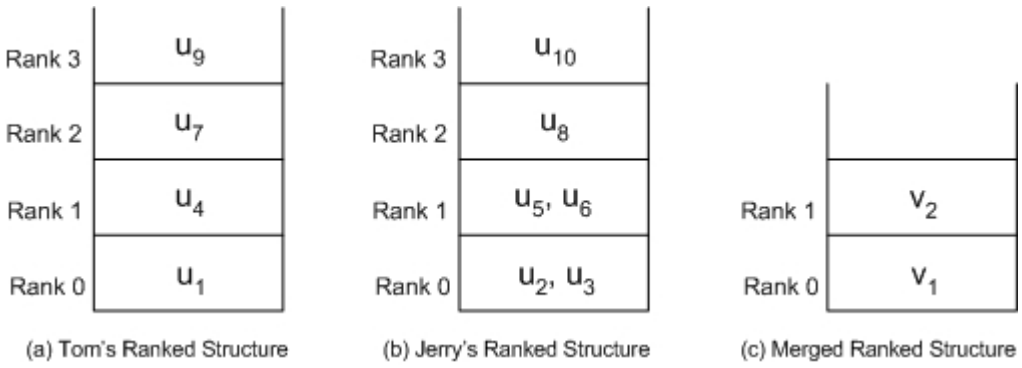

Figure 5.3: Ranked Structures

5.1.3 Summary

The study elicited 10 models involving up to 4 preference ranks from two stakeholders to finally get an agreed model that satisfied the system properties, and was conducted over a period of two days. The main effort was in eliciting models from the stakeholders and translating the agreed model in to the SMV language for model checking. The rest i.e. the consistency checking and model merging were completely done by the tool. Testing of the system properties for the agreed model using the SMV model checker was very straightforward once the model was translated into the SMV language.

In this process, at every rank when there was conflict. Both stakeholders had to relax their viewpoints gradually in order to reach an agreement. Take Tom as example. His second model (u_4 in Figure 5.1(4)) had more function allowed compared with his most preferred model (u_1 in Figure 5.1(1)) because it added a transition from state **Idle** to state **DialTone**. Although Tom's next preferred model u_7 removed this transition since it was concerned with making call instead of receiving call, he added a transition from state **Connected** to **Idle**.

A program can be used to relax viewpoints according the the relaxation policy provided by the stakeholders so that stakeholders do not continuously and repeatedly interact with the tool. Once the most preferred models and the relaxation policies are provided, the whole process can be completed by the tool.

5.2 Student Application System Case Study

In order to give a more specific idea of the iterated process of this system, we have undertaken another case study. The case study was conducted between Peter and his colleagues based on the student application process.

The case study conducted here is different from the previous one in that it involved a new stakeholder after the successful completion of the merging process between two stakeholders. The required process, i.e. the student application for submission process was expressed in finite state machines. The whole process was conducted using the tool developed in this research.

5.2.1 The Scenario

The viewpoints involved described the process of a student's application for admission. The process starts when a student makes an application, which can be accepted or rejected. If she is accepted for admission, she should confirm whether she accepts the offer or not.

5.2.2 Experiment Description

This experiment was first conducted by two stakeholders, Peter and John. For simplicity, we assume that all the stakeholders use the same vocabularies. Therefore, no vocabulary mapping was required in this case study.

First Peter started by giving his most preferred viewpoint as shown on Figure 5.4(1)(model u_1). He assumed that if a student withdrew his offer, he had to reapply, with a transition from state S_4 where variables *active* and *approve* were true and *accept* was false, to state S_1 where variables *active*, *approve* and *accept* were all false. His colleague John only specified that if a student's application was rejected, he was allowed to reapply (See model u_2 on Figure 5.4(2)). There was a transition from state $S_2$where variable *approve* was true to state S_1 where variables *active*, *approve* and *accept* were all false.

Both Peter and John had the same boolean variables: *active*, *approve* and *accept*. Variable *active* refers to the activation of the application. *approve* refers to the approval of the application and *accept* refers to the acceptance of the offer. After the viewpoints were entered into the tool, it returned a message saying that no consistent models were found.

The stakeholders then had to present their next preferred models which were models at rank 1. Peter entered his model u_3 as shown on Figure 5.4(3). He revised his model by allowing a student to reapply if his application was rejected, adding a transition from state S_2 to S_1. John changed his viewpoint by allowing a student to reaccept the offer after his withdrawal (see model u_4 on Figure 5.4(4)), by adding a transition from S_4 to S_3 where variables *active*, *approve* and *accept* were all true.

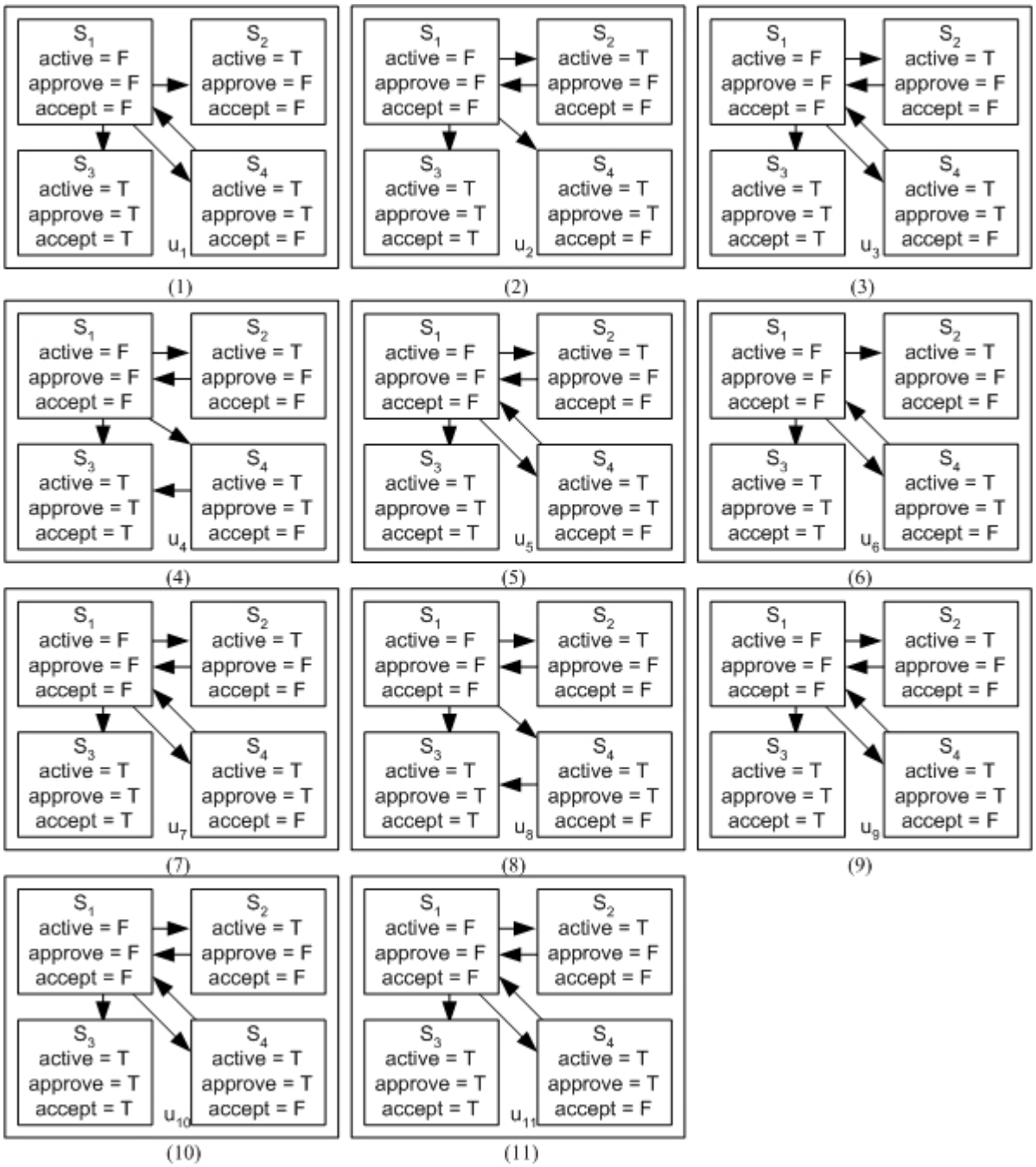

Figure 5.4: All Viewpoints Elicited for Student Application System

Now Peter's ranked structure Φ_1 contained two models u_1 and u_3 at rank 0 and rank 1 respectively. John's ranked structure Φ_2 contained models u_2 and u_4 at rank 0 and rank 1 respectively. Still no consistency was found up to this rank.

Peter and John then presented their models of next preference rank, i.e. rank 2. Peter's model u_5 was changed to allow a student to reaccept his offer if he denied the offer, the student did not have to reapply. He removed the transition from state S_4 to state S_1, and added a transition from state S_4 to state S_3 (see model u_5 on Figure 5.4(5)). John changed his viewpoint by not allowing a student to reaccept his offer if he denied it . The student would have to reapply(by changing the transition between state S_4 and state S_3 to a transition between state S_4 and state S_1). He also changed his mind that if a student's application was rejected, he could not reapply, by removing the transition from state S_2 to state S_1(see model u_6 on Figure 5.4.(6)). Now the tool checked models of $\Phi_1(u_1,\ u_3,\ u_5)$ and models of $\Phi_2(u_2,\ u_4,\ u_6)$ for consistency and detected that consistent models were found and merging operation was therefore performed (in this experiment, both Peter and John agreed to use Δ_Σ merging operator) and the results were displayed in another window showing that two merged models were found. One was ranked 0 formed by combination of model model u_1 of Peter at his preference rank 0 and model u_6 of John at his preference rank 2, the other was ranked 1 formed by combination of model u_5 of Peter at his preference rank 1 and model u_4 of John at his preference rank 1.

Actually models u_1 and u_6 were identical and models u_5 and u_4 were identical too as both Peter and John had the same states and same variables. We renamed the first combined model as v_1 and the second one as v_2. Now the newly created merged ranked structure Φ contained models v_1 and v_2. See Figure 5.6(1) for a pictorial view

of the merged ranked structure.

Models v_1 and v_2 were then model checked against the following property set by Peter and John using SMV.

Property 1 If a student denies his offer, he may have to reapply. It is represented in CTL as: AG((approve $\wedge$ $\sim$accept) $\rightarrow$ EX($\sim$active)).

The most preferred model v_1 was checked against the above property and the SMV model checker returned a *true* value, indicating model v_1 was a satisfactory model. Therefore, model checking on model v_2 was not necessary. The final output were model v_1 and the ranked structure Φ. The merging operation was deemed complete.

But at this moment, a new stakeholder Mark joined this process. He was allowed to present his most preferred model. It was model u_7 as shown on Figure 5.4(7). He allowed a student to reapply if his application was rejected. He assumed that if a student withdrew his offer, he would have to reapply. Mark's model u_7 was checked with most preferred model v_1 in the merged ranked structure Φ for consistency.

The tool detected that model u_7 was not consistent with model v_1. Therefore, Mark specified his next preferred model which was model u_8 at rank 1. See Figure 5.4(8). He changed his viewpoint by allowing a student to reaccept his offer if he withdrew it. Model v_2 at rank 1 of the merged ranked structure Φ was also taken to be compared with the models of Mark's. That is models of $\Phi(v_1, v_2)$ and models of $\Phi_3(u_7, u_8)$ were checked for consistency. The tool detected that model u_8 was consistent with model v_2. Therefore, the three stakeholders reached an agreement. Actually it was models u_5, u_4 and u_8 that were consistent. The combined model was renamed as v_3.

Now the merged ranked structure Φ was changed and contained only model v_3, which was model checked against the above described system property. The SMV model checker returned *false* for the property. Therefore, no satisfactory model was found and the merging process should continue. In this case, models of preference rank higher than the rank where Peter and John reached an agreement should be elicited from all of the three stakeholders, i.e. Peter, John and Mark. As Peter and John previously reached an agreement at rank 2, models of rank 3 were required from them.

Peter gave his model u_9 as shown on Figure 5.4(9). He assumed that if a student's application was rejected, he could reapply. If a student denied his offer, he would have to reapply too. John entered his model u_{10}. He also assumed that if a student was rejected, he could reapply. If a student denied his offer, he had to reapply. See Figure 5.4(10). Mark presented his next preferred model u_{11} (see Figure 5.4(11)) which was at rank 3 and he did not have model at rank 2 in this case. Mark assumed that if a student's application was rejected, he was not allowed to reapply. If a student to withdrew his offer he would have to reapply. Now models of Peter's ranked structure $\Phi_1(u_1, u_3, u_5, u_9)$, models of John's ranked structure $\Phi_2(u_2, u_4, u_6, u_{10})$ and models of Mark's ranked structure $\Phi_3(u_7, u_8, u_{11})$(See Figure 5.5(1), (2) and (3) for a pictorial view of the ranked structures respectively) were checked for consistency.

The tool output results showing that a model combined from consistent models u_5, u_4 and u_8 was ranked 0 (renamed as m_1), a model combined from models u_1, u_6 and u_{11} was ranked 1 (renamed as m_2), and a model combined from models u_9, u_{10} and u_7 was ranked 2 (renamed as m_3). Therefore, the merged ranked structure contained models m_1, m_2 and m_3 in the order of preference rank. (See Figure 5.6(2)

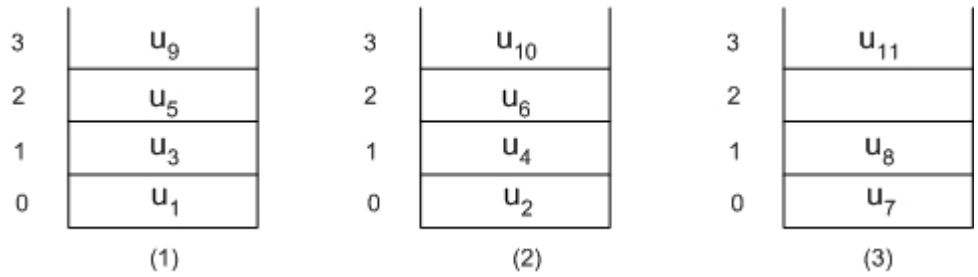

Figure 5.5: Ranked Structures of Individual Stakeholders

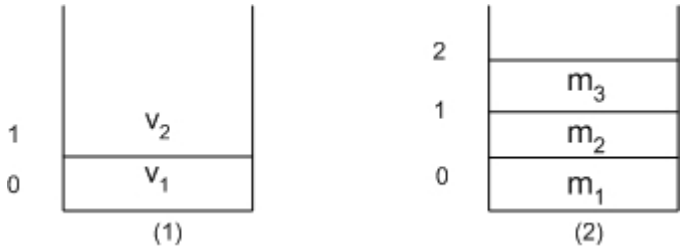

Figure 5.6: Merged Ranked Structures using Δ_{Σ}

for a pictorial view of the merged ranked structure Φ using merging operator Δ_{Σ}).

Now the most preferred model m_1 was first model checked against the system property. After running the SMV model checker, model m_1 was found not satisfying the property. Therefore, the next preferred model m_2 was sent for model checking and the SMV model checker returned *true* for the property specified. Model m_2 become the resultant output model and no further model checking was required on model m_3. Finally, the three stakeholders reached an agreement at rank 3 with output model m_2 and a merged ranked structure Φ.

In order to test the merging operation outcome with different merging operators, we run the above case study again with merging operators Δ_{min} and Δ_{max} as well using the tool. We only changed the merging operators, others retained unchanged.

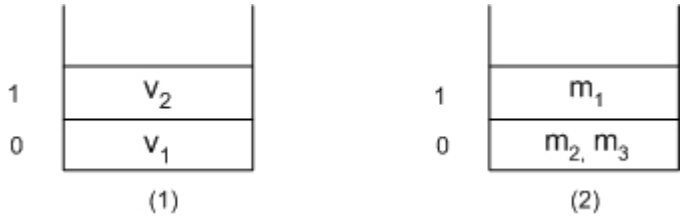

Figure 5.7: Merged Ranked Structures using Δ_{min}

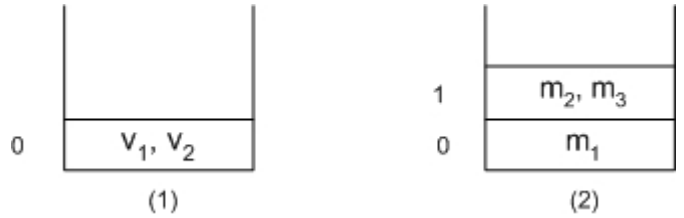

Figure 5.8: Merged Ranked Structures using Δ_{max}

The outputs were shown on Figure 5.7 (for Δ_{min} merging operator) and 5.8 (for Δ_{max} merging operator). Figures 5.7(1) and 5.8(1) are the first output resulted from the agreement between Peter and John. Figures 5.7(2) and 5.8(2) are the final outcomes (with a model satisfying the system property) agreed by all of the three stakeholders.

5.2.3 Summary

The study embodied the merging process with iteration. It elicitated 11 models up to 4 preference ranks from three stakeholders to finally get an agreed model that satisfied the system property. Consistency checking and model merging were completely done by the tool.

In this case study, which was different from the previous one that it was iterated through more stakeholders joining, different outputs were produced when more

stakeholders joined the system.

Different outputs were also produced when using different merging operators. For the output between Peter and John, the outcomes resulting from the use of Δ_{Σ} and Δ_{min} merging operators were the same, i.e. they produced the same outcome model v_1 and the same ranked structure. But for the outcome resulting from Δ_{max} merging operator was different. Although model v_1 was still the outcome model, but the ranked structure was different. See Figures 5.6(1), 5.7(1) and 5.8(1) for a pictorial view of the ranked structures using different merging operators.

The outcome produced among the three stakeholders were different too. Ranked structures produced were different. From the output ranked structure produced using merging operator Δ_{min}, if model m_3 was sent for model checking and then it would become the output model as it also satisfied the system property. It was the same for the results produced from using merging operator Δ_{max}.

5.3 Discussion

This chapter presented two different case studies on the use of merging operation to resolve inconsistent viewpoints. The results are very encouraging: belief merging was very valuable in resolving inconsistent requirements. The iterated merging process we have developed based on the idea of belief merging was extremely valuable as it allowed further merge to be performed on the already merged model. Therefore, it satisfied the iterative characteristic of requirements engineering.

The study also tested our tool used in the merging process. The tool was proven

to be helpful in automating both consistency checking and consistent model combination.

Chapter 6

Conclusion and Future Work

This project has focused on the design and development of a framework for merging multiple viewpoints in requirements engineering, which is a practical, complicated and challenging problem in the field of requirements engineering. The approach we use in this research involves the use of belief merging operations to merge multiple viewpoints.

Several belief merging operations have been proposed in the literature to solve inconsistency handling problems in requirements engineering. However some merging operations such as the one proposed by Chechik and Easterbrook, as well as the ones based on merging knowledge bases, can not specifically or sufficiently represent the behavior of the system or suggest merged outcomes.

We emphasize in this research that belief merging operations are useful in reconciling multiple stakeholder perspectives in requirements engineering and can be the basis for viewpoints merging. We have proposed the idea of ranked structures that are developed based on epistemic states. The use of epistemic states involves the provision of a complete mapping of all possible models to ranks, while ranked structures only map the currently elicited models to ranks. The models themselves differ from

those used in the belief merging framework of Meyer, since they are defined relative to local vocabularies (while Meyer's approach assumes a common vocabulary across all agents).

The framework we propose also includes the SMV model checker to check against system properties expressed in CTL. The main feature of this approach is iterative merging of multiple viewpoints from different stakeholders until a satisfactory viewpoint model is found. It also provides real-time merging operation by allowing merging newly added viewpoints from new stakeholders and hence is very flexible.

To demonstrate the applicability of the approach, we have designed and implemented a prototype incorporating the merging operators proposed by [33]to handle multiple/inconsistent requirements perspectives represented in finite state models.

As specifications used in this framework are represented in finite state models, which are formal method, it will be hard for the analysts to develop and hence being one of the limitations to this research. We are therefore motivated to refine our framework in the future.

It is observed in the case studies that relaxation policy is feasible and helpful in the elicitation process. However, we have not addressed the guidelines for forming such relaxation policy. It is anticipated to develop guidelines for relaxation policy in the future and incorporate the relaxation policy in our tool.

In this research, we only focus on handling the inconsistent requirements; the system properties have not been addressed. We consider in the future to improve our framework so that it can handle system properties (whether consistent or not), as well.

Some functionality has not been provided by our prototype. In the future, we hope to refine the prototype by implementing more functionalities. We also hope to be able to display the models graphically rather than by text. There are more merging operators proposed in [33] and we will seek to implement all of these merging operators in our tool in the future. The most important future task is considered to be the implementation of the automatic model checking using SMV model checker is feasible. As we explained in previous chapter, we need to convert the models represented in finite state models into SMV program representation so that the models can be input directly to the SMV model checker for model checking. Therefore, a device for the translation needs to be developed in the future.

Bibliography

[1] K. J. Arrow. *Social choice theory and individual values (2nd edition).* Wiley, New York, 1963.

[2] R. Balzer. Tolerating inconsistency. In *In Proceedings of the 13th Int'l Conference on Software Engineering*, pages 158–165, 1991.

[3] C. Baral, S. Kraus, and J. Minker. Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):208–220, 1991.

[4] Barry Boehm and Hoh In. Aids for identifying conflicts among quality requirements. *IEEE Software*, March 1996.

[5] B. Boehm and R. Ross. Theory w software project management:principles and examples. *IEEE Transaction on Software Engineering*, pages 902–916, July 1989.

[6] L. Bolc and P. Borowik. Many-valued logics, 1992. Springer-Verlag.

[7] L. Chung, B. A. Nixon, and E. Yu. Using non-functional requirements to systematically support change. In *In Proceedings of he Second IEEE International Symposium on Requirements Engineering*, pages 132–139, York, England, March 1995.

[8] Edmund M. Clarke, Jr. Orna Grumberg, and Doron A. Peled. *Model Checking.* Cambridge, Massachusetts.

[9] P. Gärdenfors C. E. Alchourron and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[10] D. Duffy, C. MacNish, J. McDermid, and P Morris. A framework for requirements analysis using automated reasoning. In *Proc. Seventh Advanced Conference on Information Systems Engineering*, Springer-Verlag, 1995. CAiSE*95. Lecture Notes in Computer Science.

[11] Steve Easterbrook and Marsha Chechik. A framework for multi-valued reasoning over inconsistency viewpoints. In *In Proceedings of International Conference on Software Engineering (ICSE'01)*, pages 411–420, May 2001.

[12] S. M. Easterbrook, A. C. W. Finkelstein, J. Kramer, and B. A. Nuseibeh. Coordinating conflicting viewpoints by managing inconsistency. In *Workshop on Conflict Management in Design, International Conference on Artificial Intelligence in Design*, Lausanne, Switzerland, August 1994.

[13] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Coordinating distributed viewpoints: The anatomy of a consistency check. *Int. Journal of Concurrent Engineering: Research & Applications*, 2(3):209–222, 1994.

[14] S. Easterbrook and B. Nuseibeh. Managing inconsistencies in an evolving specification. In *Proc. RE'95: Second IEEE International Symposium on Requirements Engineering*, pages 48–55, York UK, March 1995. IEEE Computer Society Press.

[15] S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *IEEE Software Engineering Journal*, pages 31–43, November 1995.

[16] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. *Transactions on Software Engineering*, 20(8):569–578, August 1994. IEEE CS Press.

[17] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and & M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–58, 1992.

[18] P. Gärdenfors. *Knowledge in flux: Modeling the Dynamics of Epistemic States*. MIT Press, 1988.

[19] A. K. Ghose. A formal basis for consistency, evolution and rationale management in requirements engineering. In *In Proceedings of the 1999 IEEE International Conference on Tools for AI*, pages 77–84. IEEE Computer Society Press, 1999.

[20] A. K. Ghose. Formal tools for managing inconsistency and change in RE. In *Proceedings of the 10th International Workshop on Software Specification and Design (IWSSD 2000)*, pages 171–182, San Diego, November 2000. IEEE Computer Society Press.

[21] "C. L. Heitmeyer, R. D. Jeffords, JOURNAL = "ACM Transaction of Software Engineering B. G. Labaw", TITLE = "Automated consistency checking of requirements specifications", and volume = "5" number = "3" pages = "231-261" Methodology", YEAR = "1996".

[22] A. Hunter and B. Nuseibeh. Managing inconsistent specifications: Reasoning, analysis and action. Technical report, Department of Computing, Imperial College, London, UK, June 1995. Technical report.

[23] A. Hunter. Measuring inconsistency in knowledge via quasi-classical models. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'2002)*, pages 68–73, 2002.

[24] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, November 1999.

[25] C. Kakas, R. A. Kowalski, and F. Toni. *The Role of Abduction in Logic Programming. Handbook of Logic in Artificial Intelligence and Logic Programming*, chapter 5, pages 235–324. Oxford University Press, 1998.

[26] R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, 1976.

[27] Sebastien Konieczny and Ramon Pino-Perez. On the logic of merging. In L. Schubert In A. G. Cohn and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, pages 488–498, San Francisco, California, 1998.

[28] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Journal of Artificial Intelligence*, 44(1-2):167–207, July 1990.

[29] Mingjune Lee and Barry Boehm. The winwin requirements negotiation system: a model-driven approach.

[30] Paolo Liberatore and Marco Schaerf. Arbitration (or how to merge knowledge bases). *IEEE Transactions on Knowledge and Engineering*, 10(1):76–90, January/February 1998.

[31] J. Lin and A. Mendelzon. Knowledge base merging by majority, 1994. Jinxin Lin and Alberto O. Mendelzon. Knowledge base merging by majority.Manuscript, 1994.

[32] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic, 1993.

[33] Thomas Meyer. On the semantics of combination operations. Journal of Applied Non-Classical Logics, 2001.

[34] T. A. Meyer, A. K. Ghose, and S. Chopra. Syntactic representations of semantic merging operations. In *Proceedings of the IJCAI-2001 Workshop on Inconsistency in Data and Knowledge*, pages 36–42, August 2001.

[35] K. Narayanaswamy and N. Goldman. "lazy" consistency: A basis for cooperative development software development. In *In Proceedings of International Conference on Computer-Supported Cooperative Work (CSCW '92)*, pages 257–264, 1992. ACM SIGCHI and SIGOIS.

[36] B. Nuseibeh and S. M. Easterbrook. The process of inconsistency management: a framework for understanding. In *In Proceedings of the First International Workshop on the Requirements Engineering process (REP 99)*, page 364, Florence, Italy, 1999.

[37] B. Nuseibeh, J. Kramer, and A. C. W. Finkeltein. Expressing the relationships between multiple views in requirements specification. In *In Proceedings of the 15th International Conference on Software Engineering (ICSE-93)*, pages 187–200, Bltimore, May 1993. IEEE Computer Society Press.

[38] B. Nuseibeh and A. Russo. Using abduction to evolve inconsistent requirements specifications. In *Proc. of ICSE99 workshop on Software Change and Evolution (1999)*, 1999.

[39] L. Perrussel and P. Charrel. Inconsistent requirements: an argumentation view. In *ICRE 2000*, September 1999.

[40] P. Z. Revesz. On the semantics of theory changes: arbitration between old and new information. In *In Proceedings PODS '93, 12th ACM SIGACT SIGMOD SIGART Symposium on the Principles of Database System*, pages 71–82, 1993.

[41] WN Robinson and S. Volkov. Conflict-oriented requirements restructuring. Gsucis working paper, Georgia State University, Atlanta, GA, April 1999.

[42] W.N. Robinson. Requirement conflict restructuring. Gsu cis working paper, Georgia State University, Atlanta, GA, 1999.

[43] K. Satoh. Consistency management in software engineering by abduction. In *Proceedings of the ICSE-2000 Workshop on Intelligent Software Engineering*, pages 90–99, Limerick, Ireland, 2000.

[44] R. W. Schwanke and G. E. Kaiser. Living with inconsistency in large systems. In *In Proceedings of International Workshop on Software Version and Configuration Control*, pages 98–118, Grassau, Germany, January 1988.

[45] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Comm. of the ACM*, 34(10):110–120, 1991.

[46] Ian Sommerville and Pete Sawyer. *Requirements Engineering, A Good Practice Guide*. John Wiley and Sons Ltd, England, 1997.

[47] V.S. Subrahmanian. On the semantics of quantitative logic programs. In *Proc. 4th IEEE Symposium on Logic Programming*, pages 173–182, Wahsington DC, 1987. Computer Society Press.

[48] V. S. Subrahmanian. Amalgamating knowledge bases. *AGM Transactions on Database systems*, 19(2):291–331, 1994.

[49] A. van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Trans. on Software Engineering*, 24(11):908–926, 1998.

[50] J. Yen and W. Tiao. A systematic tradeoff analysis for conflicting imprecise requirements. In *In Proceedings of the Third IEEE International Symposium on Requirements Engineering (RE'97)*, pages 87–96, Annapolis, Maryland, USA, January 1997.

[51] D. Zowghi, A. K. Ghose, and P. Peppas. A framework for reasoning about requirements evolution. In *Proceeding of Fourth Pacific Rim International Conference on Artificial Intelligence*, pages 157–168, Cairns, Australia, August 1996. PRICAI-96.

Appendix A

Source Code

The followings are the java source code for the implementation of the tool developed in this research. Documents are listed in the alphabetical order of the java file names.

A.1 AddArcPanel.java

```
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

/**
 * Title: AddArcPanel.java
 * Description: The program receives nodes details from stakeholders and
     write them to
                                    Access based database
 * @author Qiuming Lin
 * September 2002
 */

class AddArcPanel extends JPanel implements ActionListener
{
    static final int SIZE = 10;
    SHPanel jp_sh;
        Header header;
    JTextField jt_label;
    JComboBox jcb_inNode, jcb_outNode;
    JButton jb_add, jb_remove, jb_return;
    JList jlist_arc;
    Model m;
    Arc arc;
```

```
public AddArcPanel(){
    super();
            try {
                    jpInit();
            }
            catch(Exception e) {
                    e.printStackTrace();
            }
}

    private void jpInit() throws Exception{
            header = new Header();
            add(header);

    m = new Model();
    arc = new Arc();

    JLabel jl_title = new JLabel("Add Transition");
    jl_title.setFont(new Font("Arial",Font.BOLD,20));
    jl_title.setPreferredSize(new Dimension(ShowFrame.formX,30));
    jl_title.setHorizontalAlignment(SwingConstants.CENTER);
    add(jl_title);

        jcb_inNode = Utility.getNode();
    jcb_outNode = Utility.getNode();
    JPanel jp_header = new JPanel();
    jp_header.setLayout(new GridLayout(0,4,5,0));
    jp_header.add(ShowFrame.newLabel("In Node:"));
    jp_header.add(jcb_inNode);
    jp_header.add(ShowFrame.newLabel("Out Node:"));
    jp_header.add(jcb_outNode);
    add(jp_header);

    //display the arc added
    jlist_arc = new JList();
    JScrollPane jsp = new JScrollPane(jlist_arc);
    jsp.setPreferredSize(new Dimension(ShowFrame.formX-150,ShowFrame.
        buttonY*2));
    jsp.setBorder(BorderFactory.createTitledBorder(
                    BorderFactory.createEtchedBorder(),"Arcs Added"));
    add(jsp);

    jb_add = new JButton("Add");
    jb_add.addActionListener(this);
    jb_remove = new JButton("Remove");
    jb_remove.addActionListener(this);
    JPanel jp_button = new JPanel();
    jp_button.setLayout(new GridLayout(0,1,20,10));
    jp_button.add(jb_add);
    jp_button.add(jb_remove);
    jp_button.setPreferredSize(new Dimension(100,ShowFrame.buttonY+20));
    add(jp_button);

    jb_return = new JButton("Return");
    jb_return.addActionListener(this);
    JPanel jp_arcButton = new JPanel();
    jp_arcButton.add(jb_return);
```

```
        jp_arcButton.setPreferredSize(new Dimension(ShowFrame.formX,
            ShowFrame.buttonY));
        add(jp_arcButton);
        }

    public void actionPerformed(ActionEvent e){
        if (e.getSource()==jb_add){
            addArc();
        }else if (e.getSource()==jb_remove){
            removeArc();
        }else if (e.getSource()==jb_return){
            this.setVisible(false);
            ShowFrame.showAddModelFrame();
        }
    }

    //add an arc to model
    void addArc(){
                arc = new Arc();
        arc.setPreNode((Node)Utility.model.node.get(jcb_inNode.
            getSelectedIndex()));
        arc.setNextNode((Node)Utility.model.node.get(jcb_outNode.
            getSelectedIndex()));
        Utility.model.addArc(arc);
        printArc();
    }

    void removeArc(){
                if (jlist_arc.getSelectedIndex()>=0) {
                        Utility.model.arc.remove(jlist_arc.getSelectedIndex
                            ());
                        printArc();
                }else
                        JOptionPane.showMessageDialog(this, "Please␣select␣
                            the␣arc␣to␣be␣removed!");

    }

    void printArc(){
        Vector v = new Vector();
        for(int k=0; k<Utility.model.arc.size(); k++)
            v.add((Arc)Utility.model.arc.get(k));
        jlist_arc.setListData(v);
    }
}
```

A.2 AddModelPanel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/**
 * Title: AddModelPanel.java
 * Description: The program receives models details from stakeholders and
     write them to
```

```
   Access based database
 * @author Qiuming Lin
 * September 2002
 */

class SHPanel extends JPanel
{
    JTextField jt_shid, jt_shname;
    public SHPanel() {
        setLayout(new GridLayout(0,4,5,0));
        jt_shid = new JTextField(10);
        jt_shid.setEnabled(false);
        jt_shname = new JTextField(10);
        jt_shname.setEnabled(false);
        JLabel jl = new JLabel("Stakeholder␣ID:");
        jl.setHorizontalAlignment(SwingConstants.RIGHT);
        add(jl);
        add(jt_shid);
        jl = new JLabel("Stakeholder␣Name:");
        jl.setHorizontalAlignment(SwingConstants.RIGHT);
        add(jl);
        add(jt_shname);
        setPreferredSize(new Dimension(ShowFrame.formX,ShowFrame.fieldY));
    }
}
/*****************ADD MODEL PANEL************************************/
class  AddModelPanel extends JPanel implements ActionListener
{
        static  int SHCount = 2;
        static  int newSHCount = 1;
    JLabel jl_title, jl_rank, jl_modelName, jl_modelId;
    JTextField jt_rank,jt_modelName, jt_modelId;
    JButton jb_saveModel,jb_addNode, jb_addArc, jb_return;
        Header header;
    SHPanel jp_sh;
    JScrollPane jsp_model;

    //constructor
    public AddModelPanel() {
                try {
                        jpInit();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
    }

        private void jpInit() throws Exception{
                header = new Header();
                add(header);

        JLabel jl_title = new JLabel("Add␣Model");
        jl_title.setFont(new Font("Serif",Font.BOLD,20));
        jl_title.setPreferredSize(new Dimension(ShowFrame.formX,30));
        jl_title.setHorizontalAlignment(SwingConstants.CENTER);
        add(jl_title);

        jp_sh = new SHPanel();
```

```
jp_sh.jt_shid.setText(Utility.shid);
Database.connectDatabase();
jp_sh.jt_shname.setText(Database.getSHName(Utility.shid));
add(jp_sh);

JPanel jp_header = new JPanel();
jl_modelId = new JLabel("Model␣ID:␣");
jl_modelId.setHorizontalAlignment(SwingConstants.RIGHT);
        jt_modelId = new JTextField(getModelId());
jt_modelId.setEnabled(false);
jp_header.add(jl_modelId);
jp_header.add(jt_modelId);

jp_header.setLayout(new GridLayout(4,2));
jl_rank = new JLabel("Preference␣rank:␣");
jl_rank.setHorizontalAlignment(SwingConstants.RIGHT);
jt_rank = new JTextField(15);
        jt_rank.setEnabled(false);
        setRank();
jp_header.add(jl_rank);
jp_header.add(jt_rank);
jl_modelName = new JLabel("Model␣Name:␣");
jl_modelName.setHorizontalAlignment(SwingConstants.RIGHT);
jt_modelName = new JTextField(15);
jp_header.add(jl_modelName);
jp_header.add(jt_modelName);
add(jp_header);

JPanel jp_button = new JPanel();
jp_button.setPreferredSize(new Dimension(ShowFrame.buttonX*6,
    ShowFrame.buttonY));
jb_addNode = new JButton("Add␣State");
jb_addNode.addActionListener(this);
jb_addArc = new JButton("Add␣Transition");
jb_addArc.addActionListener(this);
jb_saveModel = new JButton("Save␣Model");
jb_saveModel.addActionListener(this);
jb_return = new JButton("Return");
jb_return.addActionListener(this);
jp_button.add(jb_addNode);
jp_button.add(jb_addArc);
jp_button.add(jb_saveModel);
jp_button.add(jb_return);
add(jp_button);
Database.closeDatabase(false);
}

private void setRank(){
        if (Database.firstRank(Utility.shid)) {
                jt_rank.setText("0");
        }else{
                int nextRank = Database.getNextRank(Utility.shid);
                int maxRank = Database.getMaxRank();
                if ((nextRank == maxRank)||(nextRank == maxRank+1))
                    {
                        jt_rank.setText(nextRank+"");
                }else{
                        jt_rank.setText(maxRank+"");
```

```
                }
            }
        }

    public void actionPerformed(ActionEvent e){
        if (e.getSource() == jb_addNode) {
            ShowFrame.showAddNodeFrame();
            this.setVisible(false);
        }else if (e.getSource()==jb_addArc){
            this.setVisible(false);
            ShowFrame.showAddArcFrame();
        }else if(e.getSource() == jb_saveModel){
                    saveModel();
        }else if (e.getSource()==jb_return){
            setVisible(false);
            ShowFrame.showSHMenuFrame();
        }
    }

        void saveModel(){
                if (dataIsValid()) {
                        Database.connectDatabase();
                        setModelDetails();
                        Database.insertModel();//insert data to database
                        Database.insertModelSH();
                        Database.insertLoginSH(Utility.shid, Utility.model.
                            rank, Utility.newSH);
                        performOperation();
                        Database.closeDatabase(false);
                }
        }

        void saveModelIteration(){
                if (dataIsValid()) {
                        Database.connectDatabase();
                        setModelDetails();
                        Database.insertModel();//insert data to database
                        Database.insertModelSH();
                        Database.insertLoginSH(Utility.shid, Utility.model.
                            rank, Utility.newSH);
                        performOperation();
                        Database.closeDatabase(false);
                }
        }

        boolean dataIsValid(){
                if (jt_rank.getText().equals("")) {
                        JOptionPane.showMessageDialog(this, "Please␣fill␣in␣
                            the␣rank␣of␣the␣model␣!");
                        return false;
                }else if (!Utility.isDigit(jt_rank.getText())){
                        JOptionPane.showMessageDialog(this, "The␣rank␣must␣
                            be␣numerals!",
                                        "Rank␣Input␣Error", JOptionPane.
                                            ERROR_MESSAGE);
                        jt_rank.setText("");
                        return false;
                }else if (jt_modelName.getText().equals("")){
```

```
                    JOptionPane.showMessageDialog(this, "Please␣fill␣in␣
                        the␣model␣name!");
                    return false;
            }
            return true;
    }

void setModelDetails(){
    Utility.model.id = jt_modelId.getText().trim();
    Utility.model.name = jt_modelName.getText().trim();
    Utility.model.rank = jt_rank.getText().trim();
}

    void performOperation(){
            int answer = JOptionPane.showConfirmDialog(null, "Model␣is␣
                added␣successfully!␣Add␣another␣one?", "Add␣Model",
                JOptionPane.YES_NO_OPTION);//yes=0, no=1
            if (answer == 0) {
                    clearModelFields();
            }else{
                    if (Database.getSHCount().size()>SHCount) {
                            SHCount = Database.getSHCount().size();
                    }
                    Vector loginSHCount = Database.getLoginSHCount(
                        Utility.model.rank, Utility.newSH);
                    if (!Utility.newSH) {
                            if ((loginSHCount.size()==SHCount)) {
                                    checkConsistency(Utility.newSH);
                            }else{
                                    setVisible(false);
                                    ShowFrame.showSHMenuFrame();
                            }
                    }else{//new stakeholders joining
                            String maxMergedRank = Database.
                                getMaxMergedRank();
                            if (loginSHCount.size()==newSHCount) {
                                    checkConsistencyIteration(Utility.
                                        newSH);
                            }else {
                                    setVisible(false);
                                    ShowFrame.showSHMenuFrame();
                            }
                            if (Utility.model.rank.equals(maxMergedRank)
                                ) {
                                    Database.updateNewLoginSH(Utility.
                                        shid);
                            }
                    }
            }
    }
    void checkConsistency(boolean newSH)
    {
            Vector m = Utility.getAllModel();
    if (CheckModel.allConsistent(m)){
                    combineModels(newSH);
    }else{
        JOptionPane.showMessageDialog(this,"Models␣are␣not␣consistent!")
            ;
```

```
                    Utility.model = new Model();
                    ShowFrame.showLoginFrameFromLogout();
                    this.setVisible(false);
            }
}
    void checkConsistencyIteration(boolean newSH)
    {
            Vector m = Utility.getAllModelIteration(Utility.model.rank);
    if (CheckModel.allConsistent(m)){
                    combineModels(newSH);
                    Database.updateAllNewLoginSH();
    }else{
        JOptionPane.showMessageDialog(this,"Models␣are␣not␣consistent!")
            ;
                    Utility.model = new Model();
                    ShowFrame.showLoginFrameFromLogout();
                    this.setVisible(false);
            }
}
    /**Combine models and assign ranks to the combined models**/
    public void combineModels(boolean newSH)
    {
            if (!newSH) {
                    if (Database.getAllMergedId().size()!=0) {
                            Vector mergedId = Database.getAllMergedId();
                            if (mergedId.size() != 0) {
                                    for (int i=0; i<mergedId.size(); i
                                        ++) {
                                            Database.deleteModel((String
                                                )mergedId.get(i));
                                    }
                            }
                    }
            }
            Vector consId = Database.getConsId();
            Vector allConsRank = Utility.getAllConsModelRankSet();
            for (int i=0; i<consId.size(); i++) {
                    Vector v = Database.getConsModel((String)consId.get(
                        i));
                    Model combinedModel = MergeModel.mergeModel(v, newSH
                        );
                    String rank = getMergedRank((String)consId.get(i),
                        allConsRank);
                    setMergedModel(v, (String)consId.get(i), rank);
                    Database.insertMergedModel();
                    Database.deleteConsModel((String)consId.get(i));
            }
            this.setVisible(false);
            JOptionPane.showMessageDialog(this, "Merging␣is␣completed␣
                sucessfully!");
            ShowFrame.showMergeResultsFrame();
    }

    private String getModelId(){
            String s = Database.getUsedModelId();
            String modelId = "";
            if(s!=null){
                    modelId = s.substring(1, modelId.length());
```

```
                        return modelId;
                }else{
                        return (Database.createModelId()+1)+"";
                }
        }

    public void setMergedModel(Vector v, String consId, String rank)
        {
        String s = "Models ";
        for (int i=0; i<v.size(); i++){
            s += ((Model)v.get(i)).id;
            if (i<v.size()-1)
                s += ", ";
        }
                Utility.model.consId = consId;
                Utility.model.rank = rank;
                Utility.model.operator = Utility.OP;
        Utility.model.desc = s;
    }
        static String getMergedRank(String consId, Vector allConsRank)
        {
                Vector consRank = Utility.getConsModelRankSet(consId);
                int mergedRank = -1;
                if (Utility.OP.equals("Delta Min")) {
                        mergedRank = MergeOperator.deltaMin(consRank,
                            allConsRank);
                }else if (Utility.OP.equals("Delta Max")){
                        mergedRank = MergeOperator.deltaMax(consRank,
                            allConsRank);
                }else if (Utility.OP.equals("Delta Sigma")){
                        mergedRank = MergeOperator.deltaSigma(consRank,
                            allConsRank);
                }
                return (mergedRank+"");
        }

    void clearModelFields()
        {
        jt_rank.setText("");
        jt_modelName.setText("");
        Utility.model = new Model(Database.createModelId()+"");
        jt_modelId.setText(Utility.model.id);
    }

    void showModel(){
        JTextArea temp = new JTextArea(300,300);
        temp.append(Utility.model.toString());
        jsp_model = new JScrollPane(temp);
        JFrame jf = new JFrame("Model");
        jf.getContentPane().add(jsp_model);
        jf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        jf.setSize(ShowFrame.formX,300);
        jf.setVisible(true);
    }
}
```

A.3 AddNodePanel.java

```
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

/**
 * Title:AddNodePanel.java
 * Description: The program receives nodes details from stakeholders and
     write them to
                                    Access based database
 * @author Qiuming Lin
 * September 2002
 */
class AddNodePanel extends JPanel implements ActionListener
{
    SHPanel jp_sh;
    JComboBox jcb_nodeName, jcb_varName, jcb_value;
    JButton jb_add,jb_remove,jb_save, jb_return;
    JList jl_variable;
        Header header;

    Node node;
    JFrame parent;

    //constructor
    public AddNodePanel(){
        super();
                try {
                        jpInit();
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
    }

        private void jpInit() throws Exception{
                header = new Header();
                add(header);

                node = new Node();

        JLabel jl_title = new JLabel("Add␣State");
        jl_title.setFont(new Font("Arial",Font.BOLD,20));
        jl_title.setPreferredSize(new Dimension(ShowFrame.formX,30));
        jl_title.setHorizontalAlignment(SwingConstants.CENTER);

        add(jl_title);

        JPanel jp_state = new JPanel();
        Database.connectDatabase();
                Vector nodeName = Database.getSHNodeName(Utility.shid);
                jcb_nodeName = new JComboBox(nodeName);
        jp_state.add(ShowFrame.newLabel("State␣Name:"));
        jp_state.add(jcb_nodeName);
```

```
        add(jp_state);

        JPanel jp_var = new JPanel();
        jp_var.setLayout(new GridLayout(1,4,3,5));
                Vector varName = Database.getSHVarName(Utility.shid);
                jcb_varName = new JComboBox(varName);
                String[] value = {"True", "False"};
                jcb_value = new JComboBox(value);

                jp_var.add(ShowFrame.newLabel("Variable: "));
                jp_var.add(jcb_varName);
                jp_var.add(ShowFrame.newLabel("Value: "));
                jp_var.add(jcb_value);

                add(jp_var);

        jl_variable = new JList();
        JScrollPane jsp = new JScrollPane(jl_variable);
        jsp.setPreferredSize(new Dimension(ShowFrame.formX-200,ShowFrame.
            buttonY*2));
        jsp.setBorder(BorderFactory.createTitledBorder(
                        BorderFactory.createEtchedBorder(),"Variables Added"
                            ));
        add(jsp);

        JPanel jp_addVarButton = new JPanel();
        jp_addVarButton.setLayout(new GridLayout(0,1,20,10));

        jb_add = new JButton("Add");
        jb_add.addActionListener(this);
        jb_remove = new JButton("Remove");
        jb_remove.addActionListener(this);

        jp_addVarButton.add(jb_add);
        jp_addVarButton.add(jb_remove);
        jp_addVarButton.setPreferredSize(new Dimension(100,ShowFrame.buttonY
            +20));

        add(jp_addVarButton);

        JPanel jp_addNodeButton = new JPanel();
        jb_save = new JButton("Save State");
        jb_save.addActionListener(this);
        jb_return = new JButton("Return");
        jb_return.addActionListener(this);
        jp_addNodeButton.add(jb_save);
        jp_addNodeButton.add(jb_return);
        jp_addNodeButton.setPreferredSize(new Dimension(ShowFrame.formX,
            ShowFrame.buttonY));

        add(jp_addNodeButton);

        Database.closeDatabase(false);
        }

    public void actionPerformed(ActionEvent e){
        if(e.getSource()==jb_add){
            setVar();//set ncde variables
```

```
    }else if(e.getSource()==jb_remove){
        removeVar();
    }else if(e.getSource()==jb_save){//add node data to model
        setNode();
        clearTextArea();
    }else if(e.getSource()==jb_return){
        ShowFrame.showAddModelFrame();
        this.setVisible(false);
    }
}
//set node variables
private void setVar(){
            if (!jcb_nodeName.getSelectedItem().equals("")) {
                    String varName = (String)jcb_varName.getSelectedItem
                        ();
                    if (!varName.equals("")) {
                            String varId = Database.getId("varVocabulary
                                ", Utility.shid, varName);
                            Variable variable = new Variable(varName,
                                varId);
                            String mid = Database.getVarMid(varId);
                            variable.setMid(mid);//System.out.println("
                                var mid: "+variable.mid);
                            String value = (String)jcb_value.
                                getSelectedItem();
                            variable.setValue(value);
                            node.setVariable(variable);
                            printVariable();
                    }
            }
}

//add node to model
void setNode(){
             String nodeName = (String)jcb_nodeName.getSelectedItem();
             if (!nodeName.equals("")) {
                     node.setName(nodeName);
                     String nodeId = Database.getId("nodeVocabulary",
                         Utility.shid, nodeName);
                     node.setId(nodeId);
                     String mid = Database.getNodeMid(nodeId);
                     node.setMid(mid);
                     Utility.model.addNode(node);
                     node = new Node();
             }
}

    void clearTextArea(){
    Vector v = new Vector();
    for(int k=0; k<node.getVarRowCount(); k++)
        v.add("");
    jl_variable.setListData(v);
    }
void removeVar(){
    if (jl_variable.getSelectedIndex()>=0){
        node.removeVariable(jl_variable.getSelectedIndex());
        printVariable();
    }else
```

```
                        JOptionPane.showMessageDialog(this, "Please␣select␣
                            the␣variable␣to␣be␣removed!");
    }
    void printVariable(){
        Vector v = new Vector();
                for (int i=0; i<node.getVarRowCount(); i++) {

                        String s = "";
                        Variable var = (Variable)node.getVariable().get(i);
                        if (var.value.equals("True")) {
                                s += var.name + "␣=␣T";
                        }else
                                s += var.name + "␣=␣F";
                        v.add(s);
                }
        jl_variable.setListData(v);
    }
}
```

A.4 Arc.java

```
import java.util.*;

/**
 * Title: Arc.java
 * Description: The program describes the arc between two nodes
 * @author Qiuming Lin
 * August/September 2002
 */

class Arc
{
    Node preNode, nextNode;
    String preNodeId, nextNodeId;

    //constructor
    public Arc()
        {
                preNodeId = "";
                nextNodeId = "";
        }

        public Arc(Arc a)
    {
        this.preNode = new Node(a.preNode);
        this.nextNode = new Node(a.nextNode);
    }
    //pre node
    public void setPreNode(Node n)
    {
        preNode = n;
    }
    public Node getPreNode(){return preNode;}

    //next node
    public void setNextNode(Node n)
    {
```

```
        nextNode = n;
    }
    public Node getNextNode(){return nextNode;}

    //check if the arc has the same pre node and next node
    public boolean isIdentical(Arc a)
    {
        if (!this.getPreNode().getMid().equals(a.getPreNode().getMid()))
            return false;
        else if (!this.getNextNode().getMid().equals(a.getNextNode().getMid
            ()))
            return false;
        else
            return true;
    }

    public String toString()
    {
                                return ("\nTransition␣"+"is␣from␣"+ this.
                                    getPreNode().name +
                "("+this.getPreNode().mid+")"+"␣to␣"+ this.getNextNode().
                    name +
                "("+this.getNextNode().mid+")");
    }

}
```

A.5 CheckModel.java

```
import java.util.*;

/**
 * Title: CheckModel.java
 * Description: The program check whether two models represented in STD are
     consistent or identical
 * @author Qiuming Lin
 * August/September 2002
 */

class CheckModel
{
    public static int consId=1;
    public static int identId=1;

        /**check all the stakeholders models of the same level**/
        public static boolean allConsistent(Vector m)
        {
                boolean cons = false;
        for (int i=0; i<m.size(); i++){
                        if (consistent((Vector)m.get(i))) {
                                Utility.insertConsModel(consId+"", (Vector)m
                                    .get(i));
                                consId++;
                                cons = true;
                        }
        }
```

```
    if (cons)
        return true;
            else
                    return false;
}
    public static boolean consistent(Vector m)
    {
            for (int i=0; i<m.size(); i++){
                    for (int j=i; j<m.size(); j++) {
                            if (!consistent((Model)m.get(i), (Model)m.
                                get(j))) {
                                    return false;
                            }
                    }
            }
            return true;
    }
/**check two models for both transition consistency and variable
    cosistency**/
public static boolean consistent(Model m1, Model m2)
{
    for(int i=0; i<m1.getNodeCount(); i++) {
        if(transitionConflict(m1,m2)) {
            return false;
        }else if (nodeConflict(m1, m2))
            return false;
    }

    return true;
}

/**transition checking**/
static boolean transitionConflict(Model m1, Model m2)
{
    boolean identical;
    Arc temp1;

    //compare mode11's transition with model2's
    for (int i=0; i<m1.getArcCount(); i++) {
        identical = false;
        temp1 = m1.getArc(i);
        if (Utility.findNode(temp1.getPreNode(), m2)!=null&&
            Utility.findNode(temp1.getNextNode(), m2)!=null){
            for (int j=0;j<m2.getArcCount(); j++)   {
                if (temp1.isIdentical(m2.getArc(j))){ //check whether
                    the two arcs have the same label,
                    identical = true;                //same pre node
                        and next node
                    break;
                }
            }
            if (!identical)
                return true;
        }
    }
    //compare model2's transition with model1's
    for (int i=0; i<m2.getArcCount(); i++){
        identical = false;
```

```
        temp1 = m2.getArc(i);
        if (Utility.findNode(temp1.getPreNode(), m1)!=null&&
            Utility.findNode(temp1.getNextNode(), m1)!=null){
            for (int j=0;j<m1.getArcCount(); j++)   {
                if (temp1.isIdentical(m1.getArc(j))){
                    identical = true;
                    break;
                }
            }
            if (!identical)
                return true;
        }
    }
    return false;
}

/**Variables Checking**/
static boolean nodeConflict(Model m1, Model m2)//compare all the nodes
    for consistent variables
{
    Node temp1, temp2;

    for (int i=0; i<m1.getNodeCount(); i++){
        temp1 = m1.getNode(i);
        temp2 = Utility.findNode(temp1,m2);
        if (temp2!=null)
            if (nodeConflict(temp1,temp2))
                return true;//inconsistent
    }
    return false;//consistent
}

/**compare two nodes for consistent variables**/
static boolean nodeConflict(Node n1, Node n2)
{
    Vector v1 = n1.getVariable();
    Vector v2 = n2.getVariable();
    for (int i=0; i<v1.size(); i++){
        for (int j=0; j<v2.size(); j++){
           if(varConflict((Variable)v1.get(i), (Variable)v2.get(j))){
               return true;//Conflict exists/Not consistent
            }
        }
    }
    return false;//consistent
}

/**compare one variable with another for consistency**/
    static boolean varConflict(Variable v1, Variable v2)
{
            if (v1.mid.equals(v2.mid)) {
                    if (v1.value.equals(v2.value)) {
                            return false;//NO conflict
                    }else
                            return true;//Conflict exists, i.e. Not
                                consistent
            }else
                    return false;//NO conflict exists, i.e. consistent
```

```
    }
}
```

A.6 Database.java

```
// Database.java
// September 2002

import java.sql.*;
import java.util.*;
import javax.swing.*;

class Database
{
    // instance variables
    public static String query;
    public static Connection con;
    public static Statement  stmt;
    public static ResultSet  results, nodeRS, arcRS, atomRS, varRS;
    static String SQLText;
    static Vector nv;
    static Model model;
    static Node node;
    static Arc arc;

/**static method to establish connection to database**/
    public static void connectDatabase() {
        String url   = "jdbc:odbc:MYDBMS";
        String username = "";
        String password = "";

        try {
            // Load the jdbc-odbc bridge driver
            Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
            // Load the Oracle jdbc driver
            // Class.forName ("oracle.jdbc.driver.OracleDriver");
            // Attempt to connect to a driver.
            con = DriverManager.getConnection(url, username, password);
            stmt = con.createStatement();
        } catch (SQLException ex) {
            while (ex != null) {
                System.out.println ("SQL␣Exception:␣␣" + ex.getMessage ());
                ex = ex.getNextException ();
            }
        } catch (java.lang.Exception ex) {
            ex.printStackTrace ();
        }
    }
/**static method to close database**/
    public static void closeDatabase(boolean withresults) {
        try {
            if (withresults){
                results.close();
                stmt.close();
                con.close();
            }
        } catch (SQLException ex) {
```

```
            while (ex != null) {
                System.out.println ("SQL Exception:  " + ex.getMessage ());
                ex = ex.getNextException ();
            }
        } catch (java.lang.Exception ex) {
            ex.printStackTrace ();
        }
    }
/**INSERT, UPDATE OR DELETE DATA**/
        public static void updateModelId(String modelId){
                 try {
           int ResultCode;
           ResultCode = stmt.executeUpdate(updateModelIdSQL(modelId));
       }catch (java.lang.Exception ex) {
           System.out.println( "** Error on data insert. **" );
           ex.printStackTrace ();
       }
        }
        public static void updateAllNewLoginSH()
        {
       try {
           int ResultCode;
           ResultCode = stmt.executeUpdate(updateAllNewLoginSHSQL());
       }catch (java.lang.Exception ex) {
           System.out.println( "** Error on data insert. **" );
           ex.printStackTrace ();
       }
    }/**/
        public static void updateNewLoginSH(String shid)
        {
       try {
           int ResultCode;
           ResultCode = stmt.executeUpdate(updateNewLoginSHSQL(shid));
       }catch (java.lang.Exception ex) {
           System.out.println( "** Error on data insert. **" );
           ex.printStackTrace ();
       }
    }
    public static void deleteConsModel(String consId)
        {
       try {
           int ResultCode;
           ResultCode = stmt.executeUpdate( deleteConsModelSQL(consId) );
       }catch (java.lang.Exception ex) {
           System.out.println( "** Error on data delete. **" );
           ex.printStackTrace ();
       }
    }
        public static void deleteLoginSH(){
                try {
                        int ResultCode;
                        ResultCode = stmt.executeUpdate(deleteLoginSHSQL());
                }
                catch ( java.lang.Exception ex) {
                        System.out.println("** Error on data delete. **");
                        ex.printStackTrace();
                }
        }
```

```
    public static void deleteModel(String modelId)
    {
   try {
       //connectDatabase();
       stmt = con.createStatement();
       int ResultCode;
               ResultCode = stmt.executeUpdate(deleteLoginSHSQL(Utility.
                   shid, getRank(modelId)));
       ResultCode=stmt.executeUpdate(deleteArcSQL(modelId));
               Vector nodeId = getNodeId(modelId);
               ResultCode = stmt.executeUpdate(deleteModelSHSQL(modelId)
                   );
               for (int i=0; i<nodeId.size(); i++) {
                       ResultCode=stmt.executeUpdate( deleteVarSQL((
                           String)nodeId.get(i), modelId));
                       ResultCode=stmt.executeUpdate( deleteNodeSQL((
                           String)nodeId.get(i), modelId));
               }
       ResultCode = stmt.executeUpdate( deleteModelSQL(modelId));
   }catch (java.lang.Exception ex) {
       System.out.println( "**␣Error␣on␣data␣insert.␣**␣" );
       ex.printStackTrace ();
   }
}
public static void insertConsModel(String consId, String modelId){
    try {
       stmt = con.createStatement();
       int ResultCode;
       ResultCode = stmt.executeUpdate( insertConsModelSQL(consId,
           modelId) );//insert into consModel table
   }catch (java.lang.Exception ex) {
       System.out.println( "**␣Error␣on␣data␣insert.␣**␣" );
       ex.printStackTrace ();
    }
}
    public static void insertNodeVoc(String shid, String name){
            try {
                    stmt = con.createStatement();
                    int ResultCode;
                    ResultCode = stmt.executeUpdate(insertNodeVocSQL(
                        shid, name));
            }
            catch (java.lang.Exception ex) {
       System.out.println( "**␣Error␣on␣data␣insert.␣**␣" );
       ex.printStackTrace ();
    }
    }
    public static void insertVarVoc(String shid, String name){
            try {
                    stmt = con.createStatement();
                    int ResultCode;
                    ResultCode = stmt.executeUpdate(insertVarVocSQL(shid
                        , name));
            }
            catch (java.lang.Exception ex) {
       System.out.println( "**␣Error␣on␣data␣insert.␣**␣" );
       ex.printStackTrace ();
    }
```

```
    }
    public static void insertModelSH(){
            try {
                    stmt = con.createStatement();
                    int ResultCode;
                    ResultCode = stmt.executeUpdate(insertModelSHSQL());
            }
            catch (java.lang.Exception ex) {
       System.out.println( "** Error on data insert. **" );
       ex.printStackTrace ();
    }
    }
    public static void insertSHDetails(String shid, String name, String
        address, String password)
    {
            try {
                    stmt = con.createStatement();
                    int ResultCode;
                    ResultCode = stmt.executeUpdate(insertSHDetailsSQL(
                        shid, name, address, password));
            }
            catch (java.lang.Exception ex) {
       System.out.println( "** Error on data insert. **" );
       ex.printStackTrace ();
    }
    }
    public static void insertLoginSH(String shid, String rank, boolean
        newSH){
            try {
                    stmt = con.createStatement();
                    int ResultCode;
                    ResultCode = stmt.executeUpdate(insertLoginSHSQL(
                        shid, rank, newSH));
            }
            catch (java.lang.Exception ex) {
       System.out.println( "** Error on data insert. **" );
       ex.printStackTrace ();
    }
    }

public static void insertMergedModel(){
   try {
       stmt = con.createStatement();
       int ResultCode;
       ResultCode = stmt.executeUpdate( insertMergedModelSQL() );//
           insert to mergedModel table
       for(int i=0; i<Utility.model.getNodeCount();i++){
           Node n = Utility.model.getNode(i);
           ResultCode=stmt.executeUpdate( insertNodeSQL(n) );//insert to
                node table
                       ResultCode = stmt.executeUpdate(insertNodeVocSQL(
                           "0", n.mid));
           for(int j=0; j<n.getVarRowCount();j++){
               Variable var = n.getVariable(j);
                               ResultCode=stmt.executeUpdate(
                                   insertVarSQL(var.id, var.value, n.id
                                   , Utility.model.id) );
           }
```

```
            }
            for(int i=0; i<Utility.model.getArcCount();i++){
                ResultCode=stmt.executeUpdate( insertArcSQL(Utility.model.
                    getArc(i)) );//insert to arc table
                        }
        }catch (java.lang.Exception ex) {
            System.out.println( "** Error on data insert. **" );
            ex.printStackTrace ();
        }
    }   /**/
    public static void insertModel(){
        try {
            stmt = con.createStatement();
            int ResultCode;
                                                System.out.println(
                                                    insertModelSQL());

            ResultCode = stmt.executeUpdate( insertModelSQL() );//insert to
                model table
            for(int i=0; i<Utility.model.getNodeCount();i++){
                Node n = Utility.model.getNode(i);
                ResultCode=stmt.executeUpdate( insertNodeSQL(n) );//insert
                    to node table
                for(int j=0; j<n.getVarRowCount();j++){
                                        Variable v = n.getVariable(j);
                                        ResultCode=stmt.executeUpdate(
                                            insertVarSQL(v.id, v.value, n.id
                                            , Utility.model.id) );
                }
            }
            for(int i=0; i<Utility.model.getArcCount();i++){
                ResultCode=stmt.executeUpdate( insertArcSQL(Utility.model.
                    getArc(i)) );//insert to arc table
                        }
        }catch (java.lang.Exception ex) {
            System.out.println( "** Error on data insert. **" );
            ex.printStackTrace ();
        }
    }
/***SQL STATEMENT***/
        private static String updateModelIdSQL(String modelId){
                String s = "X"+modelId;
                return "update model set modelId = '"+s+"' where modelId = '
                    "+modelId+"'";
        }
        private static String updateAllNewLoginSHSQL(){
                return "update loginSH set new = 0";
        }       /**/
        private static String updateNewLoginSHSQL(String shid){
                return "update loginSH set new = "+0+" where SHid = "+shid;
        }
    private static String deleteConsModelSQL(String consId){
        return "delete from consModel where consId = "+consId;
    }
        private static String deleteModelSQL(String modelId){
                return "delete from model where modelId = '" + modelId+"'";
        }
        private static String deleteNodeSQL(String nodeId, String modelId){
```

```
                return "delete␣from␣node␣where␣nodeId␣=␣"+ nodeId + "␣and␣
                    modelId␣=␣'"+modelId+"'";
        }
        private static String deleteVarSQL(String nodeId, String modelId){
                return "delete␣from␣variable␣where␣nodeId␣=␣" + nodeId + "␣
                    and␣modelId␣=␣'" + modelId+"'";
        }
        private static String deleteModelSHSQL(String modelId){
                return "delete␣from␣model_sh␣where␣modelId␣=␣'" + modelId+"'
                    ";
        }
        private static String deleteLoginSHSQL(){
                return "delete␣from␣loginSH";
        }
        private static String deleteLoginSHSQL(String shid, String rank)
                    {
                return "delete␣from␣loginSH␣where␣SHid␣=␣"+shid+"␣and␣rank
                    ␣=␣"+rank;
        }
        private static String deleteArcSQL(String modelId){
                return "delete␣from␣arc␣where␣modelId␣=␣'" + modelId +"'";
        }

        private static String insertConsModelSQL(String consId, String
            modelId){
           return "insert␣into␣consModel␣values␣(" +
                   consId+",␣'"+
                   modelId+"')";
    }
        private static String insertNodeVocSQL(String shid, String name){
                return "insert␣into␣nodeVocabulary␣values␣('" +
                                shid + "',␣'" +
                                name + "',␣" +
                                createNodeId() + ")";
        }
        private static String insertVarVocSQL(String shid, String name){
                return "insert␣into␣varVocabulary␣values␣('" +
                                shid + "',␣'" +
                                name + "',␣" +
                                createId("varId", "varVocabulary") + ")";
        }
        private static String insertModelSHSQL(){
                return "insert␣into␣model_sh␣values␣(" +
                           Utility.shid + ",␣" + Utility.model.id + ")";
        }
//insert details of source models
    private static String insertModelSQL(){
       return "insert␣into␣model␣values␣('" +
               createModelId()+"',␣'"+
               Utility.model.name+"',␣"+
               Utility.model.rank+",␣"+
               null+",␣"+
               null+",␣"+
                           null+")";
   }
   private static String insertNodeSQL(Node n){
       return "insert␣into␣node␣values␣("+
               n.id+",␣"+
```

```
            Utility.model.id+")";
    }
    private static String insertVarSQL(String varId, String value, String
        nodeId, String modelId){
        return "insert␣into␣variable␣values␣(" +
                varId+",␣'"+
                                value+ "',␣"+
                nodeId+ ",␣'"+
                                modelId+"')";
    }
    private static String insertArcSQL(Arc a){
        return "insert␣into␣arc␣values␣(" +
                a.preNode.id+",␣"+
                a.nextNode.id+",␣"+
                Utility.model.id+")";
    }
        private static String insertSHDetailsSQL(String shid, String name,
            String address, String password){
                return "insert␣into␣stakeholder␣values␣("+
                                shid+",␣'"+
                                name+"',␣'"+
                                address+"',␣'"+
                                password+"')";
        }
        private static String insertLoginSHSQL(String shid, String rank,
            boolean newSH){
                return "insert␣into␣loginSH␣values␣("+shid+",␣'"+rank+"',␣"+
                    newSH+")";
        }
//insert details of merged models
        private static String insertMergedModelSQL(){
           return "insert␣into␣model␣values␣(" +
                            Utility.model.id+",␣"+
                            Utility.model.name+","+
                            Utility.model.rank+",␣"+
                            Utility.model.consId+",␣'"+
                            Utility.model.operator+"',␣'"+
                            Utility.model.desc+"')";
        }
        public static String getConsistentRank(String modelId){
                try {
                        stmt = con.createStatement();
                        int ResultCode;
                        query = "select␣*␣from␣model␣where␣modelId␣=␣'"+
                            modelId+"'";
                        results = stmt.executeQuery( query );
                        if(results.next()){
                                return results.getString(3);
                        }
                        else
                                return null;
                }catch (java.lang.Exception ex) {
                        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
                        ex.printStackTrace();
                        return null;
                }
        }
    public static int getNextRank(String shid){
```

```
    try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣max(rank)␣from␣loginSH␣where␣SHid␣=␣" + shid;
        results = stmt.executeQuery( query );
        if(results.next()){
            return (Integer.parseInt(results.getString(1))+1);
        }
        else
            return -1;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return -1;
    }
 }
public static int getMaxRank(){
    try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣max(rank)␣from␣loginSH";
        results = stmt.executeQuery( query );
        if(results.next()){
            return (Integer.parseInt(results.getString(1)));
        }
        else
            return -1;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return -1;
    }
 }/* */
    public static String getMaxMergedRank(){
            try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣max(rank)␣from␣model␣where␣modelId␣>␣'1000'␣and␣
            modelId␣<␣'2000'";
        results = stmt.executeQuery( query );
        if(results.next()){
            return (results.getString(1));
        }
        else
            return null;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
    }
    public static String getId(String tableName, String shid, String
        name){
            try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣*␣from␣" + tableName + "␣where␣SHid␣=␣" + shid
             + "␣and␣name␣=␣'"+ name+"'";
```

```
        results = stmt.executeQuery( query );
        if (results.next()) {
            return results.getString(3);
        }else
            return null;
    }catch (java.lang.Exception ex) {
        System.out.println( "** Error on data query **" );
        ex.printStackTrace();
        return null;
    }
    }
public static String getNodeMid(String nodeId) {
    try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select * from nodeSignatureMap where nodeId = "+ nodeId
            ;
        results = stmt.executeQuery( query );
        if (results.next()) {
            return results.getString(2);
        }else
            return null;
    }catch (java.lang.Exception ex) {
        System.out.println( "** Error on data query **" );
        ex.printStackTrace();
        return null;
    }
}
public static String getVarMid(String varId) {
    try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select * from varSignatureMap where varId = "+ varId;
        results = stmt.executeQuery( query );
        if (results.next()) {
            return results.getString(2);
        }else
            return null;
    }catch (java.lang.Exception ex) {
        System.out.println( "** Error on data query **" );
        ex.printStackTrace();
        return null;
    }
}
public static String getRank(String modelId){
    try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select rank from model where modelId = '"+modelId + "'"
            ;
        results = stmt.executeQuery( query );
        if(results.next()){
            return results.getString(1);
        }
        else
            return null;
    }catch (java.lang.Exception ex) {
        System.out.println( "** Error on data query **" );
```

```
        ex.printStackTrace();
        return null;
    }
}
    public static String getMergedrank(String modelId){
            try {
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣distinct␣rank␣from␣model␣where␣modelId␣=␣'"+
            modelId +"'";
        results = stmt.executeQuery( query );
        if(results.next()){
            return results.getString(1);
        }
        else
            return null;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
    }
public static Vector getConsModel(String consId){//get one set of
    consistent models
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "select␣modelId␣from␣consModel␣where␣consId␣="+consId;
        ResultSet modelRS = stmt.executeQuery(query);
        while (modelRS.next()){
            v.add(getModel( modelRS.getString(1)));
        }
        return v;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
public static Vector getConsModelId(String consId){//get one set of
    consistent modelId
    ShowFrame.jta_process.append("geting␣consistent␣model␣ID...\n");
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "select␣modelId␣from␣consModel␣where␣consId␣=␣"+consId;
        results = stmt.executeQuery(query);
        while (results.next()){
            v.add(results.getString(1));
        }
        return v;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
public static Vector getConsId(){
```

```
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "select␣distinct␣consId␣from␣consModel";
        results = stmt.executeQuery(query);
        while (results.next()){
            v.add(results.getString(1));
        }
        return v;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
    static Vector getAllMergedId(){
            Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "select␣modelId␣from␣model␣where␣modelId␣>␣'1000'␣and␣
            modelId␣<>␣'2000'";
        results = stmt.executeQuery(query);
        while (results.next()){
            v.add(results.getString(1));
        }
        return v;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
    }
//data related to merged models
static Vector getMergedId(){
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "select␣modelId␣from␣model␣where␣modelId␣>␣'1000'␣and␣
            modelId␣<␣'+2000'";
        results = stmt.executeQuery(query);
        while (results.next()){
            v.add(results.getString(1));
        }
        return v;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}/**/
    static Vector getMergedId(String rank){
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "select␣modelId␣from␣model␣where␣modelId␣>␣'1000'␣and␣
            modelId␣<␣'2000'␣and␣rank␣=␣"+rank;
        results = stmt.executeQuery(query);
        while (results.next()){
```

```
                v.add(results.getString(1));
            }
            return v;
        }catch (java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
        }
    }
        static Vector getIteratedMergedId(){
        Vector v = new Vector();
        try{
            stmt = con.createStatement();
            query = "select modelId from model where modelId > '2000'";
            results = stmt.executeQuery(query);
            while (results.next()){
                v.add(results.getString(1));
            }
            return v;
        }catch (java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
        }
    }
/**data related to stakeholder details**/
    public static String getSHName(String id){
        try{
            stmt = con.createStatement();
            int ResultCode;
            query = "select SHName from stakeholder where SHid ="+id;
            results = stmt.executeQuery(query);
            if (results.next()){
                return results.getString(1);
                        }else
                return null;
        }catch (java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
        }
    }
        public static Vector getLoginSHCount(String rank, boolean newSH){
                Vector v = new Vector();
                try{
            stmt = con.createStatement();
            int ResultCode;
                        query = "select distinct SHid from loginSH where
                            rank ="+rank+" and new ="+newSH;
            results = stmt.executeQuery(query);
            while (results.next()) {
                v.add(results.getString(1));
            }
            return v;
        }catch (java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
```

```
    }
    }
    public static boolean firstRank(String shid){
            try{
       stmt = con.createStatement();
       query = "Select␣shid␣from␣loginSH␣where␣SHid␣=␣"+shid;
       results = stmt.executeQuery(query);
       if (results.next()){
                       if (results.getString(1).equals(null)) {
                               return true;
                       }else
                               return false;
       }else
                       return true;
    }catch(java.lang.Exception ex) {
       System.out.println( "**␣Error␣on␣data␣query␣**␣" );
       ex.printStackTrace();
       return true;
    }
    }
public static Vector getSHid(){
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        int ResultCode;
                    query = "select␣distinct␣SHid␣from␣model_sh";
        results = stmt.executeQuery(query);
        while (results.next()) {
            v.add(results.getString(1));
        }
        return v;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
public static Vector getLoginSHid(){
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        int ResultCode;
                    query = "select␣distinct␣SHid␣from␣loginSH";
        results = stmt.executeQuery(query);
        while (results.next()) {
            v.add(results.getString(1));
        }
        return v;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
public static Vector getNewSHid(){
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
```

```
            int ResultCode;
                        query = "select␣distinct␣SHid␣from␣loginSH␣where␣new
                            ␣=␣"+1;
            results = stmt.executeQuery(query);
            while (results.next()) {
                v.add(results.getString(1));
            }
            return v;
         }catch (java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
            return null;
         }
    }
    public static Vector getModelId(){
        Vector v = new Vector();
       try{
            stmt = con.createStatement();
            query = "Select␣*␣from␣model_sh";
            results = stmt.executeQuery(query);
            while (results.next()){
                v.add("Model␣"+results.getString(1)+"␣of␣Stakeholder␣"+
                   results.getString(2));
            }
            return v;
       }catch(java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
            return null;
       }
    }
//get name based on stakeholder id
    public static Vector getSHNodeName(String shid){
        Vector v = new Vector();
       try{
            stmt = con.createStatement();
            query = "Select␣*␣from␣nodevocabulary␣where␣SHid␣=␣"+shid;
            results = stmt.executeQuery(query);
            while (results.next()){
                v.add(results.getString(2));
            }
            return v;
       }catch(java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
            return null;
       }
    }
        public static Vector getSHVarName(String shid){
        Vector v = new Vector();
       try{
            stmt = con.createStatement();
            query = "Select␣*␣from␣varvocabulary␣where␣SHid␣=␣"+shid;
            results = stmt.executeQuery(query);
            while (results.next()){
                v.add(results.getString(2));
            }
            return v;
```

```
        }catch(java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
        }
    }
//get name based on its id
        public static String getNodeName(String nodeId){
                try{
            stmt = con.createStatement();
            query = "Select * from nodevocabulary where nodeId = "+nodeId;
            results = stmt.executeQuery(query);
            if (results.next()){
                return (results.getString(2));
            }else
                            return null;
        }catch(java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
        }
        }
        public static String getVarName(String varId){
                try{
            stmt = con.createStatement();
            query = "Select * from varvocabulary where varId = "+varId;
            results = stmt.executeQuery(query);
            if (results.next()){
                return (results.getString(2));
            }else
                            return null;
        }catch(java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
        }
        }
/********the above are methods in need**********************************/
        public static Vector getNodeId(String modelId){
                Vector v = new Vector();
       try{
            stmt = con.createStatement();
            query = "Select nodeId from node where modelId =  "+modelId +"'";
            results = stmt.executeQuery(query);
            while (results.next()){
                v.add(results.getString(1));
            }
            return v;
       }catch(java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
       }
        }

 /***GET ALL DATA OF ONE COMPLETE MODEL FROM THE DATABASE***/
    /**get all the models of a stakeholder**/
   public static Vector getSHModel(String shid){
```

```
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "Select␣modelId␣from␣model_sh␣where␣SHId␣=␣"+ shid;
        ResultSet modelRS = stmt.executeQuery(query);
        while (modelRS.next()){
                        String modelId = modelRS.getString(1);
                        v.add(getModel(modelId));
        }
        return v;
    }catch(java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
     public static Vector getSHModelId(String shid){
    Vector v = new Vector();
    try{
        stmt = con.createStatement();
        query = "Select␣modelId␣from␣model_sh␣where␣SHId␣=␣"+ shid;
        ResultSet modelRS = stmt.executeQuery(query);
        while (modelRS.next()){
                        v.add(modelRS.getString(1));
        }
        return v;
    }catch(java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
     static Model getMergedModel(String modelId){
             Model m = new Model(modelId);
             try {
                     stmt = con.createStatement();
                     int ResultCode;
                     query = "select␣*␣from␣model␣where␣modelId␣=␣'"+
                         modelId +"'";
                     results = stmt.executeQuery(query);
                     if (results.next()) {
                             m.rank = results.getString(3);
                             m.operator = results.getString(5);
                             m.desc = results.getString(6);
                             getNode(m);
                             getArc(m);
                             return m;
                     }else
                             return null;
             }
             catch (java.lang.Exception ex) {
         System.out.println( "**␣Error␣on␣data␣query␣**␣" );
         ex.printStackTrace();
         return null;
     }
     }
static Model getModel(String modelId){
    Model m = new Model(modelId);
```

```
    try{
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣*␣from␣model␣where␣modelId␣=␣'"+modelId+"'";
        results = stmt.executeQuery(query);
        if(results.next()){
            m.name = results.getString(2);
            m.rank = String.valueOf(results.getString(3));
            getNode(m);
            getArc(m);
            return m;
        }else
            return null;
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
        return null;
    }
}
static void getNode(Model m) {
    try{
        //Statement nodestmt = con.createStatement();
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣*␣from␣node␣where␣modelId␣=␣'"+m.id+"'";
        nodeRS = stmt.executeQuery(query);
        while (nodeRS.next()){
            Node node = new Node();
            node.id = nodeRS.getString(1);
            node.mid = getNodeMid(node.id);
            node.name = getNodeName(node.id);
            getVar(node, m);
            m.addNode(node);
        }
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
    }
}
static void getVar(Node n, Model m){
    try{
        stmt = con.createStatement();
        int ResultCode;
        query = "select␣*␣from␣variable␣where␣nodeId␣=␣"+n.id+"␣and␣
            modelId␣=␣'"+m.id+"'";
        varRS = stmt.executeQuery(query);
        while (varRS.next()){
                            Variable var = new Variable();
                            var.id = varRS.getString(1);
                            var.mid = getVarMid(var.id);
                            var.name = getVarName(var.id);
                            var.value = varRS.getString(2);
                            n.setVariable(var);
        }
    }catch (java.lang.Exception ex) {
        System.out.println( "**␣Error␣on␣data␣query␣**␣" );
        ex.printStackTrace();
    }
```

```
    }
    static void getArc(Model m){
        try{
            stmt = con.createStatement();
            int ResultCode;
            query = "select␣*␣from␣arc␣where␣modelId␣=␣'"+m.id+"'";
            arcRS = stmt.executeQuery(query);
            while (arcRS.next()){
                Arc a = new Arc();
                String preNodeId = String.valueOf(arcRS.getString(1));
                String nextNodeId = String.valueOf(arcRS.getString(2));
                for(int i=0; i<m.node.size(); i++){
                    if(preNodeId.equals(((Node)m.node.get(i)).id))
                        a.setPreNode((Node)m.node.get(i));
                    if(nextNodeId.equals(((Node)m.node.get(i)).id))
                        a.setNextNode((Node)m.node.get(i));
                }
                m.addArc(a);
            }
        }catch (java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
        }
    }
/***GET ALL DATA OF ONE COMPLETE MERGED MODEL FROM THE DATABASE***/
/**automatically generate id**/
    public static int createId(String id, String tableName){
        try{
            stmt = con.createStatement();
            int ResultCode;
            query = "select␣max("+id+")␣from␣"+tableName;
            results = stmt.executeQuery(query);
            if (results.next()){
                return (results.getInt(1)+1);
            }else
                return -1;
        }catch (java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
            return -1;
        }
    }

        public static String getUsedModelId(){
                try {
                        stmt = con.createStatement();
                        int ResultCode;
                        query = "select␣modelId␣from␣model";
                        results = stmt.executeQuery(query);
                        while (results.next()) {
                                if (results.getString(1).charAt(0)=='X') {
                                        return results.getString(1);
                                }
                        }
                        return null;
                }
                catch (java.lang.Exception ex) {
                        System.out.println("**␣Error␣on␣data␣query␣**");
```

```
                ex.printStackTrace();
                return null;
        }
}/**/
public static int createModelId(){
        try{
    stmt = con.createStatement();
    int ResultCode;
    query = "select␣max(modelId)␣from␣model␣where␣modelId␣<␣'1000'";
    results = stmt.executeQuery(query);
    if (results.next()){
                    return Integer.parseInt(results.getString(1))+1;
    }else
        return -1;
}catch (java.lang.Exception ex) {
    System.out.println( "**␣Error␣on␣data␣query␣**␣" );
    ex.printStackTrace();
    return -1;
}
}
public static int createMergedId(){
        try{
    stmt = con.createStatement();
    int ResultCode;
    query = "select␣max(modelId)␣from␣model␣where␣modelId␣>␣'999'␣
        and␣modelId␣<␣'2000'";
    results = stmt.executeQuery(query);
    if (results.next()){
        return (results.getInt(1)+1);
    }else
        return -1;
}catch (java.lang.Exception ex) {
    System.out.println( "**␣Error␣on␣data␣query␣**␣" );
    ex.printStackTrace();
    return -1;
}
}
public static int createIteratedMergedId(){
        try{
    stmt = con.createStatement();
    int ResultCode;
    query = "select␣max(modelId)␣from␣model␣where␣modelId␣>␣'1999'";
    results = stmt.executeQuery(query);
    if (results.next()){
        return (results.getInt(1)+1);
    }else
        return -1;
}catch (java.lang.Exception ex) {
    System.out.println( "**␣Error␣on␣data␣query␣**␣" );
    ex.printStackTrace();
    return -1;
}
}
public static int createNodeId(){
        try{
    stmt = con.createStatement();
    int ResultCode;
```

```
            query = "select␣max(nodeId)␣from␣nodeVocabulary␣where␣nodeId
                ␣<␣9000";
            results = stmt.executeQuery(query);
            if (results.next()){
                return (results.getInt(1)+1);
            }else
                return -1;
        }catch (java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
            return -1;
        }
        }
        public static int createMergedNodeId(){
                try{
            stmt = con.createStatement();
            int ResultCode;
            query = "select␣max(nodeId)␣from␣nodeVocabulary";
            results = stmt.executeQuery(query);
            if (results.next()){
                return (results.getInt(1)+1);
            }else
                return -1;
        }catch (java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
            return -1;
        }
        }
        public static int createMergedVarId(){
                try{
            stmt = con.createStatement();
            int ResultCode;
            query = "select␣max(varId)␣from␣varVocabulary␣where␣varId␣>␣9000
                ";
            results = stmt.executeQuery(query);
            if (results.next()){
                return (results.getInt(1)+1);
            }else
                return -1;
        }catch (java.lang.Exception ex) {
            System.out.println( "**␣Error␣on␣data␣query␣**␣" );
            ex.printStackTrace();
            return -1;
        }
        }
/**check correct login password**/
    public static boolean checkLogin(String query)
        {
        try{
            stmt = con.createStatement();
            int ResultCode;
                        results = stmt.executeQuery(query);
            if (results.next())
                return true;
            else
                return false;
        }catch (java.lang.Exception ex) {
```

```
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return false;
        }
    }
        public static String getPassword(String analystId)
        {
                try {
                        stmt = con.createStatement();
                        query = "select password from stakeholder where SHid
                            =" +analystId;
            int ResultCode;
                        results = stmt.executeQuery(query);
            if (results.next())
                return results.getString(2);
            else
                return null;
        }catch (java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
        }
        }

        public static Vector getSHCount(){
                Vector v = new Vector();
                try {
                        stmt = con.createStatement();
                        query = "select distinct SHid from model_sh";
                        results = stmt.executeQuery(query);
                        while (results.next()) {
                                v.add(results.getString(1));
                        }
                        return v;
                }
                catch (java.lang.Exception ex) {
            System.out.println( "** Error on data query **" );
            ex.printStackTrace();
            return null;
                }
        }/**/
}
```

A.7 DisplayModelPanel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.JTable;
/**
 * Title: DisplayModelPanel.java
 * Description: The program merges two consistent models
 * @author Qiuming Lin
 * September/October 2002
```

```
 */
class DisplayModelPanel extends JPanel implements ActionListener
{
    JButton jb_menu, jb_show, jb_delete;
        JList jlist_results;
    JScrollPane jsp_results;
        Header header;
        Object[][] tableData;
        JTable table;
        Object[][] results;

    public DisplayModelPanel()
        {
                try {
                        jpInit();
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
    }

        private void jpInit() throws Exception
        {
                header = new Header();
                add(header);

                JLabel jl_title = new JLabel("Models␣of␣Stakeholder␣"+
                    Utility.shid);
        jl_title.setFont(new Font("Serif",Font.BOLD,20));
        jl_title.setPreferredSize(new Dimension(ShowFrame.formX,20));
        jl_title.setHorizontalAlignment(SwingConstants.LEFT);
        add(jl_title);

                add(displayPanel());

        JPanel jp_button = new JPanel();
        jp_button.setPreferredSize(new Dimension(ShowFrame.formX*3,ShowFrame
            .buttonY));
        jb_menu = new JButton("Menu");
        jb_menu.addActionListener(this);
        jp_button.add(jb_menu);
                jb_show = new JButton("Show␣Model");
                jb_show.addActionListener(this);
                jp_button.add(jb_show);
        jb_delete = new JButton("Delete");
        jb_delete.addActionListener(this);
        jp_button.add(jb_delete);
        add(jp_button);
        }

    public void actionPerformed(ActionEvent e)
        {
        Database.connectDatabase();
        if (e.getSource()==jb_menu){
            setVisible(false);
            ShowFrame.showSHMenuFrame();
        }else if (e.getSource()==jb_show){
                        showModel();
```

```
        }else if (e.getSource()==jb_delete){
            deleteModel();
        }
        Database.connectDatabase();
    }
        void deleteModel()
    {
                int row = table.getSelectedRow();
                if (row>=0) {
                        Object modelId = getValueAt(row, 0);
                        Database.updateModelId((String)modelId);
                        JOptionPane.showMessageDialog(this, "Model␣"+(String
                            )modelId+"␣is␣deleted␣sucessfully!");
                 }else
                         JOptionPane.showMessageDialog(this, "Please␣select␣
                            the␣model␣to␣be␣deleted!");
        Database.connectDatabase();
    }
        public Object getValueAt(int row, int col)
        {
            return results[row][col];
    }
        public Object[][] setTable()
        {
                Vector model = Database.getSHModel(Utility.shid);
                results = new Object[model.size()][3];
                for (int i=0; i<model.size(); i++) {
                        Model m = (Model)model.get(i);
                        results[i][0] = m.id+"";
                        results[i][1] = m.name+"";
                        results[i][2] = m.rank;
                }
                return results;
        }
        public JPanel displayPanel()
        {
                JPanel jp_results = new JPanel();
                String[] columnNames = {"Model␣Id", "Model␣Name", "Rank"};
                tableData = setTable();
                table = new JTable(tableData, columnNames);
                table.setPreferredScrollableViewportSize(new Dimension
                    (500, 70));
                jsp_results = new JScrollPane(table);
                jp_results.add(jsp_results);
                return jp_results;
        }
        public void showModel()
        {
                String id = JOptionPane.showInputDialog(this, "Enter␣the␣
                    model␣ID:␣");
                Model m = Database.getModel(id);
                Utility.showModel(m.toString());
        }
}
```

A.8 Go.java

```
//by Qiuming Lin
//August 2002
/*************************************************************************
 * The program is to implement our framework of merging operation to    *
 * resolve inconsistency in requirements engineering                    *
 *************************************************************************/
//Main function

import java.util.*;

public class Go
{
    public static void main(String[] args)
        {
                ShowFrame.showLoginFrame();
    }
}
```

A.9 MergeModel.java

```
import java.util.*;

/**
 * Title: MergeModel.java
 * Description: The program merges two consistent models
 * @author Qiuming Lin
 * August/September 2002
 */

class  MergeModel
{
    static Node node = new Node();
    static Arc arc = new Arc();
    static int nodeId;
    static int arcId;

    /**merge a set of consistent models**/
    public static Model mergeModel(Vector m, boolean iterated)
        {
        Utility.model = (Model)m.get(0);
        for (int i=1; i<m.size(); i++){
            Model temp = new Model(Utility.model, Utility.model.id);
            mergeModel(temp, (Model)m.get(i), iterated);
        }
                return Utility.model;
    }

    /**merge two consistent models**/
    public static Model mergeModel(Model m1, Model m2, boolean iterated)
    {
                int id;
                if (iterated) {
                        id = Database.createIteratedMergedId();
                }else{
                        id = Database.createMergedId();
                }
                Utility.model = new Model(m2, id+"");
```

```
    for (int i=0; i<Utility.model.getArcCount(); i++){
        setNewPreNextNodeId(Utility.model.getArc(i), Utility.model);
    }
    mergeNode(m1, m2, Utility.model);
    mergeTransition(m1, m2, Utility.model);
            return Utility.model;
}

public static void mergeTransition(Model m1, Model m2, Model m)
{
    boolean identical = false;
    for (int i=0; i<m1.getArcCount(); i++){
                    identical = false;
        for (int j=0; j<m2.getArcCount(); j++){
            if (m1.getArc(i).isIdentical(m2.getArc(j)))
                                identical = true;
        }
                    if (!identical) {
                            setNewPreNextNodeId(m1.getArc(i), m);
                            m.addArc(m1.getArc(i));
                    }
    }
}

private static void setNewPreNextNodeId(Arc a, Model m)
    {
    for (int k=0; k<m.getNodeCount(); k++){
        if (a.preNode.mid.equals(m.getNode(k).mid)){
            a.preNode.id = m.getNode(k).id;
                            a.preNode.mid = m.getNode(k).mid;
                    }
        if (a.nextNode.mid.equals(m.getNode(k).mid)){
            a.nextNode.id = m.getNode(k).id;
                            a.nextNode.mid = m.getNode(k).mid;
                    }
    }
}

public static void mergeNode(Model m1, Model m2, Model m)
{
    Vector v;
    Node nodeFound;
    for (int i=0; i<m1.getNodeCount(); i++) {
        nodeFound = Utility.findNode(m1.getNode(i),m2);
        if (nodeFound==null){//add node to model m
            m1.getNode(i).id = String.valueOf(nodeId++);
            m.addNode(m1.getNode(i));
        }else{//merge variables and set the merged variables to the
            merged model m
            v = mergeVariable(m1.getNode(i), nodeFound);
            for (int j=0; j<m.getNodeCount(); j++){//set merged
                variables to the node in the merged model m
                if (m.getNode(j).getMid().equals(nodeFound.getMid())){
                                        m.getNode(j).getVariable().
                                            clear();
                    for(int k=0;k<v.size();k++){
                        m.getNode(j).setVariable(new Variable((Variable)
                            v.get(k)));
```

```
                                                }
                        break;
                    }
                }
            }
    }
}

/**merge variables of two consistent nodes**/
public static Vector mergeVariable(Node n1, Node n2)
{
    boolean remove;
            boolean add;
            boolean conflict;
    Vector v1 = n1.getVariable();
    Vector v2 = n2.getVariable();
            Vector v = new Vector();

            for (int i=0; i<v2.size(); i++) {//copy v2 to v
                    v.add(new Variable((Variable)v2.get(i)));
            }
    for (int i=0; i<v1.size(); i++){//merge v1 and v2
        remove = false;
                    add = true;
        for (int j=0; j<v2.size(); j++){
                            if (((Variable)v1.get(i)).mid.equals(((
                                Variable)v2.get(j)).mid)) {
                                    add = false;
                            }else{
                                    if (CheckModel.varConflict((Variable
                                        )v1.get(i), (Variable)v2.get(j))
                                        ) {
                                            add = false;
                                            remove((Variable)v2.get(j),
                                                v);
                                    }
                            }
        }
        if (add){
                            for (int j=0; j<v1.size(); j++) {
                                    v.add(new Variable((Variable)v1.get(
                                        j)));
                            }
                    }
            }
    return v;
}

    public static void add(Variable v1, Vector v2)
    {
            Vector temp = new Vector();
            Variable v = new Variable(v1);
            v2.add(v);
    }

    public static void remove(Variable v1, Vector v2)
    {
            for (int i=0; i<v2.size(); i++) {
```

```
                        if (v1.mid.equals(((Variable)v2.get(i)).mid)) {
                                v2.remove(v2.get(i));
                                i--;
                                break;
                        }
                }
        }
}
```

A.10 MergeOperator.java

```
import java.util.*;

/**
 * Title: MergeOperator.java
 * Description: The program merges two consistent models
 * @author Qiuming Lin
 * September/August 2002
 */

class  MergeOperator
{
    public static int minRank(Vector rank){//find the minimum rank among
        vector rank
        int minRank=Integer.parseInt((String)rank.get(0));
        for (int i=1; i<rank.size(); i++){
            if (Integer.parseInt((String)rank.get(i))<minRank)
                minRank = Integer.parseInt((String)rank.get(i));
        }
        return minRank;
    }
/**DELTA MIN OPERATOR**/
    public static int deltaMin(Vector rank, Vector rankV){//the final rank
        return (findMinRank(rank) - minRank(minRankSet(rankV)));
    }
    /**get the minimum rank of a set of consistent models**/
    public static int findMinRank(Vector rank){
        boolean equal = true;
        int minRank=0;
        for (int i=0; i<rank.size()-1; i++){
            for (int j=i+1; j<rank.size(); j++){
                if (Integer.parseInt((String)rank.get(i))!= Integer.parseInt
                    ((String)rank.get(j))){
                    equal = false;
                    break;
                }
            }
        }
        if (equal){
            minRank = Integer.parseInt((String)rank.get(0)) * 2;
        }else{
            minRank = minRank(rank) * 2 + 1;
        }

        return minRank;
    }
```

```
        //get a set of minimum ranks of consistents of models.
    public static Vector minRankSet(Vector rank){
        Vector v = new Vector();
        for (int i=0; i<rank.size(); i++){
            v.add(findMinRank((Vector)rank.get(i))+"");
        }

        return v;
    }

/**DELTA MAX OPERATOR**/
    public static int deltaMax(Vector rank, Vector rankV){
       return findMaxRank(rank) - minRank((Vector)(maxRankSet(rankV)));
    }
    /**get the maximum rank among a set of consistent models**/
    public static int findMaxRank(Vector rank){
        int maxRank = Integer.parseInt((String)rank.get(0));
        for (int i=1; i<rank.size(); i++){
            if (Integer.parseInt((String)rank.get(i))>maxRank)
                maxRank = Integer.parseInt((String)rank.get(i));
        }

        return maxRank;
    }
        //get the maximum ranks of all sets of consistent models
    public static Vector maxRankSet(Vector rank){
        Vector v = new Vector();
        for (int i=0; i<rank.size(); i++){
            v.add(findMaxRank((Vector)rank.get(i))+"");
        }
        return v;
    }

/**DELTA SIGMA OPERATOR**/
    public static int deltaSigma(Vector rank, Vector rankV){
        return deltaSigma(rank)-minRank(allDeltaSigma(rankV));
    }
    public static int deltaSigma(Vector rank){//get sum of the a Vector rank
        int sigma = 0;
        for (int i=0; i<rank.size(); i++){
            sigma += Integer.parseInt((String)rank.get(i));
        }
        return sigma;
    }
    public static Vector allDeltaSigma(Vector rank){//get sum of the a
        Vector rank
        Vector v = new Vector();
        for (int i=0; i<rank.size(); i++){
            v.add(deltaSigma((Vector)rank.get(i))+"");
        }
        return v;
    }

/**refined delta sigma operator**/
    public static int refSigma(Vector rank){
        int refSigma=0;
                //not implemented yet
        return refSigma;
```

```
    }
}
```

A.11 MergeResultsPanel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.JTable;
/**
 * Title: MergeModelPanel.java
 * Description: The program merges two consistent models
 * @author Qiuming Lin
 * September/October 2002
 */
class MergeResultsPanel extends JPanel implements ActionListener
{
    JButton jb_menu, jb_show, jb_exit, jb_modelCheck;
    JScrollPane jsp_tableData;
        Header header;
        Object[][] tableData;
        JTable table;

    public MergeResultsPanel()
        {
                try {
                        jpInit();
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
    }

        private void jpInit() throws Exception
        {
                header = new Header();
                add(header);

                JLabel jl_title = new JLabel("Results␣of␣Merge");
        jl_title.setFont(new Font("Serif",Font.BOLD,20));
        jl_title.setPreferredSize(new Dimension(ShowFrame.formX,20));
        jl_title.setHorizontalAlignment(SwingConstants.CENTER);
        add(jl_title);

                add(resultPanel());

        JPanel jp_button = new JPanel();
        jp_button.setPreferredSize(new Dimension(ShowFrame.formX*3,ShowFrame
            .buttonY));
        jb_menu = new JButton("Menu");
        jb_menu.addActionListener(this);
        jp_button.add(jb_menu);
                jb_show = new JButton("Show␣Model");
                jb_show.addActionListener(this);
```

```
            jp_button.add(jb_show);
            jb_modelCheck = new JButton("Model Check");
            jb_modelCheck.addActionListener(this);
            jp_button.add(jb_modelCheck);
        jb_exit = new JButton("Exit");
        jb_exit.addActionListener(this);
        jp_button.add(jb_exit);
        add(jp_button);
        }

    public void actionPerformed(ActionEvent e)
        {
        if (e.getSource()==jb_menu){
            setVisible(false);
            ShowFrame.showSHMenuFrame();
        }else if (e.getSource()==jb_show){
                        showModel();
        }else if (e.getSource()==jb_modelCheck){
                        System.out.println("Not implemented yet");
        }else if (e.getSource()==jb_exit){
            Database.closeDatabase(true);
                        this.setVisible(false);
            System.exit(1);
        }
    }
        public Object[][] setTable()
        {
                Database.connectDatabase();
                Vector mergedId;
                if (Utility.newSH) {
                        mergedId = Database.getIteratedMergedId();
                }else
                        mergedId = Database.getMergedId();
                Object[][] tableData = new Object[mergedId.size()][4];
                for (int i=0; i<mergedId.size(); i++) {
                        Model m = Database.getMergedModel((String)mergedId.
                            get(i));
                        tableData[i][0] = m.rank+"";
                        tableData[i][1] = m.id+"";
                        tableData[i][2] = m.operator;
                        tableData[i][3] = m.desc;
                }
        Database.closeDatabase(false);
                return tableData;
        }

        public JPanel resultPanel()
        {
                JPanel jp_tableData = new JPanel();
                String[] columnNames = {"Rank", "Model Id", "Operator", "
                    Description"};
                tableData = setTable();
                table = new JTable(tableData, columnNames);
                table.setPreferredScrollableViewportSize(new Dimension
                    (500, 70));
                jsp_tableData = new JScrollPane(table);
                jp_tableData.add(jsp_tableData);
                return jp_tableData;
```

```
        }

        public void modelCheck()
        {
                //not implemented yet
        }
        public void showModel()
        {
                String id = JOptionPane.showInputDialog(this, "Enter␣the␣
                    model␣ID:␣");
        Database.connectDatabase();
                Model m = Database.getMergedModel(id);
                Utility.showModel(m.toString());
        Database.closeDatabase(false);
        }
}
```

A.12 Model.java

```
import java.util.*;

/**
 * Title: Model.java
 * Description: The program describes a model represented in a state
     transition diagram
 * @author Qiuming Lin
 * August/September 2002
 */

class Model
{
    public String id, name, rank, consId, desc, operator;//,mergedId, shid
    Vector node;
    Vector arc;

//constructor
    public Model(){
                id =  "";
                name = "";
                rank = "";
                consId = "";
                desc = "";
                operator = "";
        node = new Vector(1,1);
        arc = new Vector(1,1);
    }
    public Model(String i){  //initialize constructor
        this();
        id = i;
    }
    public Model(Model m, String id){//copy constructor
        this.id = id;
                node = new Vector();
                arc = new Vector();
        for (int i=0; i<m.getNodeCount(); i++){
            Node temp = new Node(m.getNode(i));
```

```
                    node.add(temp);
        }
        for (int j=0; j<m.getArcCount(); j++){
            Arc temp = new Arc(m.getArc(j));
                    arc.add(temp);
        }
    }
    public String getModelId() { return id; }

//node
    public void addNode(Node n){
        node.add(n);
    }
    public void removeNode(Node n) {}
    public Node getNode(int i) { return (Node)node.get(i); }
    public int getNodeCount(){ return node.size(); }

//arc
    public void addArc(Arc a){
        arc.add(a);
    }
    public void removeArc(Arc a){}
    public Arc getArc(int i){ return (Arc)arc.get(i);}
    public int getArcCount(){return arc.size();}

    public void setrank(String r){
        rank = r;
    }
    public String getrank(){return rank;}

//display model
    public String toString(){
        String s = "\n\nModel␣Name:␣"+name+"\tModel␣ID:␣"+id;
       /* if(node != null){
            s += "\n\nIt has following nodes";
            for (int i=0; i<node.size(); i++){
                s += node.get(i);
            }
        }
        if(arc != null){
            s += "\n\nIt has following transtion: ";
            for (int i=0; i<arc.size(); i++){
                s += arc.get(i);
                        }
        }*/
        return s;
    }

}
```

A.13 MyJFrame.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
```

```
 * Title: MyJFrame.java
 * Description: The program is mainly to set the frame in the center of the
    screen
 * @author Qiuming Lin
 * August/September 2003
 */

public class MyJFrame extends JFrame
{

        public MyJFrame()
        {
                addWindowListener(new CloseWindow());
        }

        public MyJFrame(String title)
        {
                super(title);
                addWindowListener(new CloseWindow());
        }

        public void center()
        {
                Dimension screenSize = Toolkit.getDefaultToolkit().
                   getScreenSize();
                int screenWidth = screenSize.width;
                int screenHeight = screenSize.height;

                Dimension frameSize = this.getSize();
                int x = (screenWidth - frameSize.width)/2;
                int y = (screenHeight - frameSize.height)/2;

                if (x<0){
                        x = 0;
                        frameSize.width = screenWidth;
                }

                if (y<0){
                        y = 0;
                        frameSize.height = screenHeight;
                }
                this.setLocation(x,y);
        }
}

class CloseWindow extends WindowAdapter
{
        public void windowClosing (WindowEvent e){
                System.out.println("closing window");
                System.exit(0);
        }
}
```

A.14 Node.java

```
import java.util.*;
```

```
/**
 * Title: Node.java
 * Description: The program describes a node, which is a state of a state
     transition diagram
 * @author Qiuming Lin
 * August/September 2002
 */
/*********************************CLASS VARIABLE
    ********************************/
class Variable
{
    String id, name, value;
    String mid;

        public Variable(){
                id = "0";
                name = "unknown";
                value = "unknown";
        }

    public Variable(String n){name=n;}

    public Variable(String na, String id){
                name = na;
                this.id = id;
        }
    public Variable(Variable v){
                id = new String(v.id);
        name = new String(v.name);
        mid = new String(v.mid);
                value = new String(v.value);
    }
    public void setMid(String m){ mid = m;}
    public String getMid(){return mid;}

        public void setValue(String v){value = v;}
        public String getValue(){return value;}

        public String toString(){
                return ("\nVariable␣Id:␣"+id +
                                "\tVariable␣Name:␣" + name +
                                "\tMid␣value:␣" + mid +
                                "\tVariable␣Value:␣" + value);
        }
}

/*********************************CLASS NODE*******************************
    */
class Node
{
    String id, name, mid;
    Vector variable;

    public Node(){
                id = "";
                name = "";
                mid = "";
        variable = new Vector();
```

```
}

public Node(String na, String id){
    name = na;
    this.id = id;
}
    public Node(Node node, String name)
    {
            id = node.id;
            this.name = name;
            mid = node.mid;
            variable = new Vector();
        for (int i=0; i<node.variable.size(); i++) {
                Variable temp = (Variable)node.variable.get(i);
                    variable.add(temp);
        }
    }
public Node(Node node){
   id = node.id;
       name = node.name;
       mid = node.mid;
       variable = new Vector();
       for (int i=0; i<node.variable.size(); i++) {
                Variable temp = (Variable)node.variable.get(i);
                    variable.add(temp);
       }
}
    public void setId(String id){this.id = id;}
    public String getId(){return id;}
public void setName(String na){name = na;}
public String getName(){return name;}

/**mid node's name**/
public void setMid(String m) {mid = m;}
public String getMid() {return mid;}

    public void setVariable(Variable v){ variable.add(v);}
public Variable getVariable(int i){ return (Variable)variable.get(i);}
public Vector getVariable(){ return variable;}
    public void removeVariable(int i){      variable.removeElementAt(i)
        ; }
public int getVarRowCount() { return variable.size(); }

public boolean isIdentical(Node n){
    Vector v1 = this.getVariable();
    Vector v2 = n.getVariable();

    if (v1.size()!=v2.size())
        return false;
    else{
        for (int i=0; i<v1.size(); i++){
            boolean identical = false;
            for (int j=0; j<v2.size(); j++) {
                if (Utility.sameVar((Vector)v1.get(i), (Vector)v2.get(j)
                    ))
                    identical = true;
            }
            if (!identical) {
```

```
                    return false;
                }
            }
        }
        return true;
    }
    /**display node**/
    public String toString(){
        return ("\n\nNode␣name:␣"+name+"\tNode␣ID:s␣"+this.id+"\tMid␣value:␣
            "+getMid()+
                                "\n\nIts␣variables␣are:\n"+variable);
    }
}
```

A.15 RegistrationPanel.java

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;

/**
 * Title: RegistrationPanel.java
 * Description: The program receives details of stakeholders and write them
     to
   Access based database
 * @author Qiuming Lin
 * August 2003
 */

/******************ADD MODEL PANEL************************************/

class  RegistrationPanel extends JPanel implements ActionListener
{
        static final int SIZE = 10;
    JLabel jl_state, jl_var, jl_id, jl_name, jl_address, jl_password;
    JTextField jt_id,jt_name, jt_address;
        JPasswordField jt_password;
    JButton jb_submit, jb_reset;
        Header header;
        JTextField jt_state[], jt_var[];
    JScrollPane jsp_state, jsp_var;

    //constructor
    public RegistrationPanel()
        {
                try {
                        jpInit();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
    }

        private void jpInit() throws Exception
        {
```

```
        header = new Header();
        add(header);

JLabel jl_title = new JLabel("Stakeholder␣Registration");
jl_title.setFont(new Font("Serif",Font.BOLD,20));
jl_title.setPreferredSize(new Dimension(ShowFrame.formX,30));
jl_title.setHorizontalAlignment(SwingConstants.CENTER);
add(jl_title);

        add(registrationPanel());

        jsp_state = new JScrollPane(stateNamePanel());
        jsp_state.setPreferredSize(new Dimension(ShowFrame.formX
            -100,ShowFrame.buttonY*2));
jsp_state.setBorder(BorderFactory.createTitledBorder(
                BorderFactory.createEtchedBorder(),"Please␣enter␣the
                    ␣state␣vocabularies"));
        add(jsp_state);

        jsp_var = new JScrollPane(varNamePanel());
        jsp_var.setPreferredSize(new Dimension(ShowFrame.formX-100,
            ShowFrame.buttonY*2));
jsp_var.setBorder(BorderFactory.createTitledBorder(
                BorderFactory.createEtchedBorder(),"Please␣enter␣the
                    ␣variable␣vocabularies"));
        add(jsp_var);

add(buttonPanel());
}

private JPanel registrationPanel(){
        JPanel jp_registration = new JPanel();
        jp_registration.setLayout(new GridLayout(4,2, 3, 5));
        jp_registration.setPreferredSize(new Dimension(ShowFrame.
            formX-100,ShowFrame.buttonY*3));
        jp_registration.setBorder(new TitledBorder(""));
        jl_id = new JLabel("Stakeholder␣ID:␣");
        jl_id.setHorizontalAlignment(SwingConstants.RIGHT);
Database.connectDatabase();
        jt_id = new JTextField(Database.createId("SHid","stakeholder
            ")+"");
        jt_id.setEnabled(false);
        jl_name = new JLabel("Stakeholder␣Name:␣");
        jl_name.setHorizontalAlignment(SwingConstants.RIGHT);
        jt_name = new JTextField(20);
        jl_address = new JLabel("Stakeholder␣Address:␣");
        jl_address.setHorizontalAlignment(SwingConstants.RIGHT);
        jt_address = new JTextField(30);
        jl_password = new JLabel("Password:␣");
        jl_password.setHorizontalAlignment(SwingConstants.RIGHT);
        jt_password = new JPasswordField(10);
        jp_registration.add(jl_id); jp_registration.add(jt_id);
        jp_registration.add(jl_name); jp_registration.add(jt_name);
        jp_registration.add(jl_address); jp_registration.add(
            jt_address);
        jp_registration.add(jl_password); jp_registration.add(
            jt_password);
Database.closeDatabase(false);
```

```
        return jp_registration;
}

private JPanel stateNamePanel()
{
        jt_state = new JTextField[SIZE];
        JPanel jp_state = new JPanel();
        jp_state.setLayout(new GridLayout(0, 2, 0, 1));
        for (int i=0; i<SIZE; i++) {
                jp_state.add(ShowFrame.newLabel("State␣"+(i+1)+":␣")
                    );
                jt_state[i] = new JTextField(10);
                jp_state.add(jt_state[i]);
        }
        return jp_state;
}

private JPanel varNamePanel()
{
        jt_var = new JTextField[SIZE];
        JPanel jp_var = new JPanel();
        jp_var.setLayout(new GridLayout(0,2,0,1));
        for (int i=0; i<SIZE; i++) {
                jp_var.add(ShowFrame.newLabel("Variable␣"+(i+1)+":␣"
                    ));
                jt_var[i] = new JTextField(10);
                jp_var.add(jt_var[i]);
        }
        return jp_var;
}

private void insertData()
{
        String id = jt_id.getText().trim();
        String name = jt_name.getText().trim();
        String address = jt_address.getText().trim();
        char[] x = jt_password.getPassword();
        String password = new String(x);
Database.connectDatabase();
        Database.insertSHDetails(id, name, address, password);
        insertVarName();
        insertStateName();
Database.closeDatabase(false);
}

private void insertStateName()
{
        int i=0;
        while (!jt_state[i].getText().trim().equals("")) {
                Database.insertNodeVoc(jt_id.getText(), jt_state[i].
                    getText());
                i++;
        }
}

private void insertVarName()
{
        for (int i=0; i<SIZE; i++) {
```

```
                        if (!jt_var[i].getText().trim().equals("")) {
                                Database.insertVarVoc(jt_id.getText(),
                                    jt_var[i].getText());
                        }
                }
        }

        private JPanel buttonPanel()
        {
                JPanel jp_button = new JPanel();
        jp_button.setPreferredSize(new Dimension(ShowFrame.buttonX*2,
            ShowFrame.buttonY));
        jb_submit = new JButton("Submit");
        jb_submit.addActionListener(this);
        jb_reset = new JButton("Reset");
        jb_reset.addActionListener(this);
        jp_button.add(jb_submit);
        jp_button.add(jb_reset);
                return jp_button;
        }

    public void actionPerformed(ActionEvent e)
        {
        if (e.getSource() == jb_submit) {
            this.setVisible(false);
                        insertData();
                        Database.closeDatabase(true);
                        this.setVisible(false);
                        ShowFrame.showLoginFrameFromLogout();
        }else if (e.getSource()==jb_reset){
                        clearField();
        }
    }

        private void clearField()
        {
                jt_name.setText("");
                jt_address.setText("");
                jt_password.setText("");
                for (int i=0; i<SIZE; i++)
                        jt_state[i].setText("");
                for (int j=0; j<SIZE; j++)
                        jt_var[j].setText("");
        }
}
```

A.16 ShowFrame.java

```
//ShowFrame.java
//Written by Qiuming Lin
//August/September 2002
//sept 19 11:37am

import java.util.*;
import javax.swing.*;
import java.awt.*;
```

```
class ShowFrame
{
   static final int formX = 500;
   static final int formY = 500;
   static final int fieldY = 22;
   static final int buttonX = 100;
   static final int buttonY = 50;
   static JTextArea jta_process = new JTextArea();
   static LoginPanel jp_login;
   static AddModelPanel jp_model;
   static AddNodePanel jp_node;
   static AddArcPanel jp_arc;
   static DisplayModelPanel jp_display;
   static SHMenuPanel jp_SHMenu;
   static MergeResultsPanel jp_mergeResults;
   static RegistrationPanel jp_registration;
   static MyJFrame frame, processFrame;


   public static JLabel newLabel(String s)      {
       JLabel jl = new JLabel(s);
       jl.setHorizontalAlignment(SwingConstants.RIGHT);
       return jl;
   }
   public static void showProcessFrame(){
       processFrame = new MyJFrame("Process");
       processFrame.getContentPane().setLayout(new GridLayout(0,1));
       processFrame.getContentPane().add(new JScrollPane(jta_process));
       processFrame.setSize(100,300);
       setProcessFrameLoc();
       processFrame.setVisible(true);
   }
   static void setProcessFrameLoc(){
       processFrame.setLocation((int)(frame.getLocation().getX()+frame.
           getSize().getWidth()),
                                (int)(frame.getLocation().getY()));
       processFrame.setSize(300,(int)frame.getSize().getHeight());
       processFrame.show();
   }
/**show frames**/
   static void newFrame(String title, int x, int y){
       frame.setTitle(title);
       frame.setSize(x,y);
           frame.center();
       //setProcessFrameLoc();
   }
   static void showLoginFrame(){
       frame = new MyJFrame("Stakeholder Login");
       frame.setSize(formX-100,350);
           frame.center();
       frame.setDefaultCloseOperation(MyJFrame.EXIT_ON_CLOSE);
           jp_login = new LoginPanel();
       frame.getContentPane().add(jp_login);
       frame.setVisible(true);
   }

   static void showLoginFrameFromLogout(){
           newFrame("Stakeholder Login", formX-100,350);
```

```
        jp_login = new LoginPanel();
        frame.getContentPane().add(jp_login);
        frame.show();
}

/* static void showAnalystFrame(){
        newFrame("DSL Merging Project - System Maintenance",formX
            -100,350);
    frame.getContentPane().add(new AnalystPanel());
    frame.show();
}*/
static void showSHMenuFrame(){
    newFrame("DSL Merging Project - Menu",formX-100,350);
        jp_SHMenu = new SHMenuPanel();
    frame.getContentPane().add(jp_SHMenu);
    frame.show();
}
static void showAddModelFrame(){
    newFrame("DSL Merging Project - Add Model",buttonX*6, 350);
    frame.getContentPane().add(jp_model);
    jp_model.setVisible(true);
    frame.show();
}
static void showAddArcFrame(){
    newFrame("Add Transition",formX+100,formY-150);
    jp_arc = new AddArcPanel();
    frame.getContentPane().add(jp_arc);
    jp_arc.setVisible(true);
    frame.show();
}
static void showAddNodeFrame(){
    newFrame("DSL Merging Project - Add State",formX+150,formY-150);
    jp_node = new AddNodePanel();
    frame.getContentPane().add(jp_node);
    jp_node.setVisible(true);
    frame.show();
}
static void showDisplayModelFrame(){
    newFrame("DSL Merging Project - Display Models",formX+100,300);
    jp_display = new DisplayModelPanel();
    frame.getContentPane().add(jp_display);
    jp_display.setVisible(true);
    frame.show();
}/**/
static void showMergeResultsFrame(){
    newFrame("DSL Merging Project - Result of Merge",formX+100,formY-150)
        ;
    jp_mergeResults = new MergeResultsPanel();
    frame.getContentPane().add(jp_mergeResults);
    jp_mergeResults.setVisible(true);
    frame.show();
 }
     static void showRegistrationForm(){
             newFrame("DSL Merging Project - Stakeholder Registration",
                 formX, formY+100);
             jp_registration = new RegistrationPanel();
             frame.getContentPane().add(jp_registration);
             jp_registration.setVisible(true);
```

```
                frame.show();
        }
        static void showAnalystMenuFrame()
        {
       newFrame("DSL␣Merging␣Project␣-␣Analyst␣Menu",formX-100,350);
       frame.getContentPane().add(new AnalystMenuPanel());
       frame.show();
        }
}
```

A.17 Utility.java

```
//Uitility.java
//Written by Qiuming Lin
//August/September 2002
//sept 19 11:37am

import java.util.*;
import javax.swing.*;

class Utility
{
    static final int SIZE=10;
        static Node node = new Node();
        static Model model = new Model();
        static Vector mergedRanking;
        static String shid = "";
        static final String OP = "Delta␣Min";
        static boolean newSH = false;

/**get data from database**/
    public static JComboBox getNode()
        {
        Vector v = new Vector();
        for(int i=0;i<Utility.model.node.size();i++){
            v.add((i+1)+".␣"+Utility.model.getNode(i).name);
        }
        if(v != null)
            return ( new JComboBox(v) );
        else
            return null;
    }
        public static JComboBox getNodeName(String shid)
        {
                Vector nodeName = Database.getSHNodeName(shid);
                for (int i=0; i<nodeName.size(); i++) {
                        nodeName.add((i+1)+".␣"+nodeName.get(i));
                }
                if (nodeName != null)
                        return (new JComboBox(nodeName));
                else
                        return null;
        }
        public static JComboBox getVarName(String shid)
        {
                Vector varName = Database.getSHVarName(shid);
                for (int i=0; i<varName.size(); i++) {
```

```
                        varName.add((i+1)+". "+varName.get(i));
                }
                if (varName != null)
                        return (new JComboBox(varName));
                else
                        return null;
        }
/*******************the above methods are in use
    *******************************/
    public static Vector getRank(Vector modelId)
        {
        Vector v = new Vector();
        for (int i=0; i<modelId.size(); i++){
           v.add(Database.getRank((String)modelId.get(i)));
        }
        return v;
    }
        public static Vector getConsModelRankSet(String consId)
        {
                Vector modelId = Database.getConsModelId(consId);
                return (getRank(modelId));
        }
    public static Vector getAllConsModelRankSet()
        {
       Vector v = new Vector();
       Vector consModelId = getConsModelId();
       for (int i=0; i<consModelId.size(); i++){
           v.add(getRank((Vector)consModelId.get(i)));
       }
       return v;
    }
    public static Vector getConsModelId()//get all sets of consistent
        modelId
        {
                Vector consM = new Vector();
        Vector consId =Database.getConsId();
        for (int i=0; i<consId.size(); i++){
            consM.add(Database.getConsModelId((String)consId.get(i)));
        }
        return consM;
    }
        public static Vector getMergedId(String rank){
                int size = Integer.parseInt(rank);
                Vector v = Database.getMergedId("0");
                if (size>0) {
                        for (int i=0; i<size+1; i++)
                                v.add(Database.getMergedId(i+""));
                }
                return v;
        }
        public static Vector getAllModel(){
                Vector shid = Database.getSHid();
                Vector modelId = getAllSHModelId(shid);
                Vector modelIdSet = concatAllModelIds(modelId);
                Vector m = getAllModel(modelIdSet);
                return m;
        }
        public static Vector getAllModelIteration(String rank){
```

```
        Vector v = new Vector();
        System.out.println("rank␣=␣"+rank);
        Vector id = getMergedId(rank);
        v.add(id);
        Vector shid = Database.getNewSHid();
        Vector modelId = getAllSHModelId(shid);
        for (int i=0; i<modelId.size(); i++) {
                v.add(modelId.get(i));
        }
        Vector modelIdSet = concatAllModelIds(v);
        Vector m = getAllModel(modelIdSet);
        return m;
}
public static Vector getModels(Vector modelId)//get one set of
    models
{
        Vector v = new Vector();
        for (int i=0; i<modelId.size(); i++) {
                v.add((Model)Database.getModel((String)modelId.get(i
                    )));
        }
        return v;
}

public static Vector getAllModel(Vector modelId)//get all sets of
    models
{
        Vector v = new Vector();
        for (int i=0; i<modelId.size(); i++) {
                v.add((Vector)getModels((Vector)modelId.get(i)));
        }
        return v;
}

public static Vector getAllSHModelId(Vector shid)//get sets of model
     id of all the stakeholders
{
        Vector v = new Vector();
        for (int i=0; i<shid.size(); i++) {
                Vector modelId = Database.getSHModelId((String)shid.
                    get(i));
                v.add(modelId);
        }
        return v;
}

public static Vector concatAllModelIds(Vector m){//input a vector of
     vector of model ID of all stakeholders
        Vector v = concatModelIds((Vector)m.get(0), (Vector)m.get(1)
            );
        if (m.size()>2) {
                Vector output = new Vector();
                for (int i=2; i<m.size(); i++) {
                        concatModelIds(v, (Vector)m.get(i), output);
                        v = output;
                }
                return output;
        }else
```

```
                return v;
        }

        public static Vector concatModelIds(Vector m1, Vector m2, Vector m)
            //m1 is a vector of vector, m2 is vector.
        {
                                                                        //
            add element of m2 to element vector of m1
                for (int i=0; i<m1.size(); i++) {
                        for (int j=0; j<m2.size(); j++) {
                                Vector v = new Vector();
                                Vector v3 = (Vector)m1.get(i);
                                for (int k=0; k<v3.size(); k++) {
                                        v.add(v3.get(k));
                                }
                                v.add(m2.get(j));
                                //System.out.println("second v = "+v);
                                m.add(v);
                        }
                }
                return m;
        }

        public static Vector concatModelIds(Vector m1, Vector m2)
        {
                Vector m = new Vector();
                for (int i=0; i<m1.size(); i++) {
                        for (int j=0; j<m2.size(); j++) {
                                Vector m3 = new Vector();
                                m3.add(m1.get(i));
                                m3.add(m2.get(j));
                                m.add(m3);
                        }
                }
                return m;
        }

        public static void insertConsModel(String consId, Vector m)
        {
                for (int i=0; i<m.size(); i++) {
                        Database.insertConsModel(consId,((Model)m.get(i)).id
                            );
                }
        }

        public static void showModel(String model)
        {
       JTextArea temp = new JTextArea(300,300);
       temp.append(model);
       JScrollPane jsp_model = new JScrollPane(temp);
       JFrame jf = new JFrame("Display␣Model");
       jf.getContentPane().add(jsp_model);
       jf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
       jf.setSize(ShowFrame.formX,300);
       jf.setVisible(true);
        }
/**check whether Node n exists in Model m**/
    static Node findNode(Node n, Model m)
```

```
        {
        for (int i=0; i<m.getNodeCount(); i++)
            if (n.getMid().equals(m.getNode(i).getMid()))
                return m.getNode(i);

        return null;
    }

        static boolean isNewSH(String loginSHid)
        {
                boolean newSH = false;
                if (Database.getMergedId().size()!=0) {
                        newSH = true;
                        Vector shid = Database.getSHid();//get shid from
                            model_sh
                        for (int i=0; i<shid.size(); i++) {
                                if (loginSHid.equals((String)shid.get(i))) {
                                        newSH = false;
                                        break;
                                }
                        }
                }
                if (newSH) {
                        return true;
                }else
                        return false;
        }
        static boolean isNewSH(){
                boolean same = true;
                Vector loginSHid = Database.getLoginSHid();
                Vector shid = Database.getSHid();//from table model_sh
                for (int i=0; i<loginSHid.size(); i++) {
                        same = true;
                        for (int j=0; j<shid.size(); j++) {
                                System.out.println("(String)shid.get(j)=␣"+(
                                    String)shid.get(j));//
                                    ////////////////////
                                if (!((String)loginSHid.get(i)).equals((
                                    String)shid.get(j))) {
                                        same = false;
                                        break;
                                }
                        }
                }
                System.out.println("same␣=␣"+same);//
                    ////////////////////////////////////////
                if (!same) {
                        return true;
                }
                return false;
        }
/**compare one variable with another for identicality**/
    public static boolean sameVar(Vector v1, Vector v2)
        {
        boolean equal=false;
        if (v1.size()!=v2.size())
            return false;
        else {
```

```
            for (int i=0; i<v1.size(); i++){
                equal = false;
                for (int j=0; j<v2.size(); j++){;
                    if (((Variable)v1.get(i)).getMid().equals(((Variable)v2.
                        get(j)).getMid())){
                         equal = true;
                                                break;
                                        }
                }
                                if (!equal)
                                        return false;
                        }
        }
        return true;
    }
    public static boolean atomOk(Vector v)
        {
        String compared;
        int j=0;
        while(true&&j<v.size()){
            compared = (String)v.get(j++);
            if (!equalAtom(compared,j, v))
                return true;
        }
        return false;
    }
    public static boolean equalAtom(String s, int j, Vector v)
        {
        for (int k=j; k<v.size(); k++)
            if (s.equals(v.get(k)))
                return true;

        return false;
    }
    public static boolean isDigit(String s)
        {
        for (int i=0; i<s.length(); i++)
            if (!java.lang.Character.isDigit(s.charAt(i)))return false;
        return true;
    }

}
```