ARCHITECTURAL ASPECTS AND A PROTOTYPE SYSTEM FOR HANDLING DISPUTES IN ELECTRONIC COMMERCE TRANSACTIONS

CENTRE FOR NEWFOUNDLAND STUDIES

TOTAL OF 10 PAGES ONLY
MAY BE XEROXED

(Without Author's Permission)

YIMING LEI

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisisitons et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-89645-5
Our file Notre référence
ISBN: 0-612-89645-5

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canadä

Architectural Aspects and a Prototype System for Handling Disputes in Electronic Commerce Transactions

by

© Yiming Lei

A thesis submitted to the
School of Graduate Studies
in partial fulfillment of the requirements
for the degree of
Master of Science

Department of Computer Science
Memorial University of Newfoundland

August 2002

St. John's Newfoundland

Abstract

In this thesis, we study some issues relating to the architecture for dispute-handling in electronic commerce (EC). We first propose a model for the dispute-handling architecture in EC transactions and describe how various components work together in a cooperative manner. We then focus our attention on a critical component, rule processing, that underlies the effective functioning of the entire system. We discuss how the notion of rules can be applied to assist players in proving propositions. Since all rules are not equally reliable, we introduce the concept of rule weight that reflects the reliability of a rule, and the algorithm for rule weight calculation. We show that the application of weak rules, i.e., rules that do not have full weights, makes it more probable to prove propositions. We indicate the problems resulting from the application of weak rules, and propose methods to cope with them. Finally, to study the practical feasibility of our architecture, we present an implementation strategy and apply it to a prototype system. The implementation follows the 3-tier client-server structure of our architectural model, and applies Java-related techniques.

Acknowledgments

I would like to thank Dr. Jian Tang, my supervisor, for his patience and support over the past three years and for continuing as my supervisor during his sabbatical leave. Without his guidance, this thesis would not have taken this final form.

Of course, I am grateful to my parents for their unconditional love. Their understanding and encouragement helped me pass through the toughest period during this thesis research.

Finally, I would like to express my gratitude to many other members in the Department of Computer Science for their kind assistance. For instance, computer system administrators provided an excellent computing environment for this research. Also, Mrs. Jane Foltz helped proofread an earlier draft of this thesis and corrected some typographical and grammatical errors.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Electronic Commerce Concepts

The word Electronic Commerce (EC) refers to business activities involving consumers, manufacturers, service providers, and intermediaries, accomplished through the use of computer networks [2,12]. There are other definitions in the literature. In general, however, the term EC implies that the business processes are conducted electronically via networks. It is generally accepted that there are two major categories of EC applications: Business-To-Consumer (B2C) and Business-To-Business (B2B) [17]. B2C is a term that stresses the direction of delivery: B2C commerce is supposedly something done by businesses to consumers. This domain is founded on intense customer focus. Example areas include web-based retail, Internet auctions, etc. On the other hand, B2B involves exchanging products and services between business organizations. Typical forms of B2B are procurement and inventory exchange, both of which encourage intercompany trading across entire industries.

EC's goal is to improve efficiency and effectiveness of the trading process by applying advanced information technologies, such as distributed computing, process automation, etc. Information technologies have developed very rapidly in recent years. As a result, EC has experienced an explosion. For instance, the increasing popularity of the Internet, a ubiquitous digital infrastructure, provides an extremely attractive new medium for EC. In the past, businesses could conduct activities with each other over closed proprietary networks, a process usually referred to as Electronic Data Interchange (EDI). EDI never gained much popularity because of its high communication costs, and requirements for specialized networks [1]. Yet the exponential growth of the Internet has changed this paradigm. Business activities can now be conducted efficiently among various participants on a global scale at low costs. The resulting explosive growth and continuing expansion of EC may be illustrated by the figures in table 1.1 from [7].

Table 1.1: Revenues for US eCommerce Goods, 1999 – 2003

Category	1999	2003 (expected)
B2B	USD 43 bn	USD 1.3 tn
B2C	USD 8 bn	USD 108 bn

The significance of EC is manifold. First, EC increases the speed and efficiency of business transactions and processes, thus improving customer services. Because EC is achieved with the help of computers, numerous tasks can be accomplished by software programs automatically and efficiently. Next, EC enhances competition and thus reduces prices for goods and services. For example, there are many travel agencies on the Internet and hence a consumer is able to compare their flight prices very easily. In such an environment, unnecessary high prices are almost impossible. Also, EC creates new services and businesses, which can lead to job creation and economic growth. Moreover, EC enables enterprises to conduct business with distant partners in the same way as they do with neighboring partners. This is because computer networks can provide highly efficient communications that make distance "disappear".

1.2 EC Disputes

Electronic commerce has a very promising future as an efficient business type. However, it does pose some problems that were rarely considered to be important before. For example, when a consumer is trying to purchase a book from the Internet, how does she know if the online book store is a true retailing business? Who can guarantee that the book will be delivered after the consumer has paid? Even if the book is delivered successfully to the consumer, it may be that the book is not what the consumer has ordered. These kinds of anomalies are almost non-existent in traditional face-to-face business activities. Yet for EC, questions like above become serious because different parties may know and contact each other only through electronic messages. In traditional forms of business, people tend to make deals only with those trusted parties. While in EC, it is hard to guarantee the trustworthiness of various participants. For instance, it is very hard for a consumer to know in advance whether a business which is behind a fancy WWW shopping site is credible or not. Also, hand written signatures have been widely used and acknowledged as legal guarantees for business agreements in traditional commerce. However, in EC field, it is not possible so far to apply a legally binding facility in electronic forms that is as convenient, popular, and cheap as a hand written signature.

The steps that participants follow to conduct commercial activities in an EC process are governed by a collection of rules. Ideally, these rules are designed in such a way that participants can mutually benefit and their interests can be protected. From above, however, these predefined rules may not always be followed faithfully by all EC participants. In such cases, disputes may arise because participants may disagree with the trading results and some participants may feel that the trading is unfair. A dispute is an argument raised by some participant, which is usually composed of a claim and a request. The claim statement states why the dispute initiator thinks the trading is unfair or unsatisfactory, e.g., the explanation on what has gone wrong in the process. And the request statement states something that the dispute initiator thinks should be done to reinstall his/her satisfaction. Look at the following example. Suppose a consumer ordered a desktop computer from an online store, and after the consumer had paid, the merchant, i.e., the store, delivered the computer system. This is a typical scenario of B2C commerce. After receiving the computer, however, the consumer found that the system did not work well, e.g., the system crashed easily. The consumer, a participant of the above EC process, hence decided to initiate a dispute. The consumer may claim 'The merchant delivered a bad computer to me' and request 'the merchant take back the bad computer and deliver another good one to me'.

Therefore, proper handling of disputes is an important topic for EC research. The American Arbitration Association [3] has pointed out: "If the upside of eCommerce is the ability to do business faster than ever before, the downside is for eCommerce-related disputes to arise even faster." Nowadays, EC is playing a more and more important role in the overall economy, and EC is also an indispensable drive for technology innovations. Consequently, studies for handling .EC disputes have significant meaning for both economy and technology developments.

However, to our best knowledge, not much work has been done in this field. Most research efforts to date focus on the generation and collection of evidence that can be used in case some participant misbehaves. Yet they usually pay little attention to procedures of dispute handling, such as how to generate a resolution, how to assist participants when they are unable to provide corresponding evidence, and what to do to improve handling efficiency.

The work in [4] proposes a framework for dispute handling, it however does not present a unified correctness criterion for decision generation. Neither does it discuss in detail how to deal with cases where some evidence is lost or withheld. Another work on EC disputes is introduced in [19], where a protocol with automated dispute resolution is proposed. Nevertheless, the protocol is only useful for exchanging digital items and thus has limited applications. A B2B EC Dispute Management Protocol has been proposed by the American Arbitration Association [3]. Yet the protocol provides only guidelines on fair dispute resolutions. The protocol indeed depends on human arbitration, although it incorporates some computer technologies, e.g., an online system which can facilitate communications between EC participants and can help locate human mediators and arbiters. In-depth investigations on EC dispute handling are presented in [23,24]. The authors model the aspects of EC transactions in such a way that some support to dispute handling can be provided. However, some important issues are missing in their work. For example, they do not present a method in which the various parts are integrated into a functional system that can work in concert. Their work has realized the significance of using rules for proposition proving. However, there is no discussion on how to evaluate rules in terms of reliabilities. Neither do they discuss on how to deal with conflicts resulting from rules with diverse reliabilities. We believe these issues are important since they are directly related to the applicability of the system in practical applications.

1.3 Contributions of the Thesis

Because EC activities usually involve participants from different geographical areas, it is very hard, if not impossible, to construct a unified legal framework for EC disputes. Online Dispute Resolution [18] argued that "in cyberspace, courts don't work very well — they're tied to geography, and cross-boundary jurisdiction can be very complicated to untangle." Hence, off-court resolution is a promising direction for handling EC disputes.

EC activities are conducted via electronic means and are based on the application of computer networks, so it is our belief that EC disputes should be handled with the help of computer systems. Such an approach can make it convenient to communicate with all involved participants. Also, the utilization of software processes can automate many procedures in the dispute handling. As a result, high efficiency can be achieved at a relatively low cost.

From above, therefore, in this thesis we study some special issues relating to EC-dispute handling. We first propose an EC dispute handling architecture. The architecture uses a client-server model and can be viewed as an off-court alternative resolution for dispute handling. In our model, servers are subdivided into tiers according to their functionalities. The first tier server is the arbitration server, which includes a software arbiter and a human arbiter [24]. During a dispute handling process, the arbitration server interacts with the second tier servers, such as rule base server and protocol tree server to retrieve necessary information. These servers are connected to the third tier servers, such as database servers, which manage and implement information by means of various data models.

The second contribution of the thesis is a framework for rule processing. When a dispute arises, the parties involved need to prove their claims. Rules are used for that purpose. We discuss in depth how the necessary rules are obtained, and what if they are not available from the protocol. We substantiate the notion of weak rules proposed in [24], and show how it is created, evaluated, and used. The main idea is to use weights for the reliabilities of weak rules. We propose algorithms for weight assignment. We also indicate the problems and pitfalls as a result of using weak rules, and propose solutions to cope with them.

The third contribution of this thesis is a strategy for the implementation. Since our architecture contains multi-tier servers, which require complex interactions between them, adoption of proper implementation scheme is vital both in effectiveness and efficiency of the system. We use Java as the implementation language for its flexibility in terms of platform independence, and RMI as the means for remote communications. We have also actually implemented partially a prototype system. (Refer to Section 5 for more detail.)

The rest of this thesis is organized as follows: Section 2 introduces basic concepts related to EC dispute handling and our arbiter architecture. This section also summarizes related works. Section 3 describes the overall arbiter architecture. Section 4 details a vital component of the architecture — rule base server and the corresponding strategies. Section 5 discusses implementation issues about the architecture. A prototype system currently under development is described as well in this section. Section 6 concludes the topic and suggests some directions for future work.

Chapter 2

Background

In this chapter, we review some concepts that will be used in the later chapters. Unless otherwise mentioned, these are proposed in [23,24].

2.1 EC Transactions

An EC **transaction** is a process for people/companies to conduct commercial activities via EC infrastructure. The participants of the transaction are called **players**. Players execute transactions by exchanging **messages.** In other words, an EC transaction is modeled as a sequence of message transmissions. Messages can be either electronic or tangible entities.

2.2 Atomicity and Transaction States

Atomicity is the property that guarantees the following: for multiple operations, either all of them are executed or none of them are. An excellent general introduction on transaction atomicity is given in [16]. The author in [9] introduces implementation details for atomic transaction processing systems.

The notion of atomicity is extended to the EC context recently by some researchers: money atomicity and goods atomicity are introduced in [26], and the purchase atomicity is proposed in [24].

Definition 2.1 *A fund transfer operation preserves money atomicity if once the customer makes a payment the merchant will receive it and vice versa.*

Definition 2.2 *A goods delivery operation preserves goods atomicity if once the merchant dispatches the goods the customer will receive it and vice versa.*

Definition 2.3 *An EC transaction preserves purchase atomicity if (1) funds transfer preserves money atomicity, (2) goods delivery preserves goods atomicity, and (3) either the order has been placed, the customer has paid according to the order and the customer gets the goods specified on the order with the exact value (quality and quantity), or none of these three things has effectively occurred.*

We adopt purchase atomicity (or simply, atomicity) as the criterion for judging whether or not the players involved in an EC transaction have traded in a fair way.

It is convenient to describe atomicity in terms of **transaction state**. To this end, we consider three abstract state variables, *order, money* and *goods*. A transaction state is simply an assignment of values to these state variables. The variable *order* takes value of one if an order has been placed, and zero otherwise, *money* is assigned the amount transferred if fund transfer is completed, and zero otherwise, and *goods* is stored with the delivered quantity of goods, and zero otherwise.

Therefore, exchanging messages among players can cause state transitions since order, money and goods are all exchanged as messages. Let s be a sequence of messaging, and Q and R be two states. We use Q →s R to denote that s causes a state transition from Q to R.

Definition 2.4 *State R preserves atomicity if Q →s R where Q is the initial state and the occurrences of messaging in sequence s preserve atomicity.*

2.3 EC Transaction Protocols

An EC transaction protocol is a collection of rules that stipulate how EC transactions should be executed. In the following, we present Open Buying on the Internet (OBI) [20] protocol as an example[1].

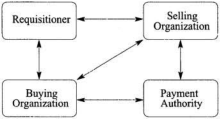


Figure 2.1: Players and Message Exchanges in OBI Protocol

OBI is an open standard protocol for B2B EC solutions. It is targeted at high-volume, low-dollar transactions that account for most of organizations' purchasing activities. Figure 2.1 illustrates the players and message exchanges in the OBI protocol.

The OBI protocol works in the following way: requisitioner is a member within buying organization and is allowed to shop selling organization's merchant server through a web browser. Requisitioner can browse an on-line catalog of goods and services and make a selection. Based on the content of requisitioner "shopping basket", selling organization forms an OBI order request and sends the request to buying organization. If buying organization approves the order request, it then creates a complete OBI order from the order request. Buying organization returns the formatted OBI order to

[1] We choose OBI here, instead of the simpler but artificial one in [24], to show the consistency of the modeling concepts with the practical applications.

selling organization. With the help of payment authority, selling organization obtains credit authorization and ships the ordered merchandise. Payment authority issues an invoice and receives payment. In some cases, e.g., for frequently ordered items, an alternative procedure may be in place. That is, buying organization sends "unsolicited" OBI order to selling organization without requisitioner's first "shopping" selling organization's catalog.

2.4 EC Protocol Trees

An EC protocol can be represented by listing all the transaction executions that follow the protocol definition. It is convenient to represent a protocol by a tree structure. A tree representing an EC protocol is called an EC **protocol tree**. Because our intention is to have a structure containing sufficient information to assist in dispute handling which usually involves abnormal transaction executions, we further require that those executions which do not follow the protocol definition should also be represented in the protocol tree as long as they are predictable.

Corresponding to the description of OBI protocol, Figure 2.2 is the OBI protocol tree. Please note: for simplicity, we do not consider the cases where buying organization sends "unsolicited" OBI order to selling organization without requisitioner's first "shopping" selling organization's catalog.

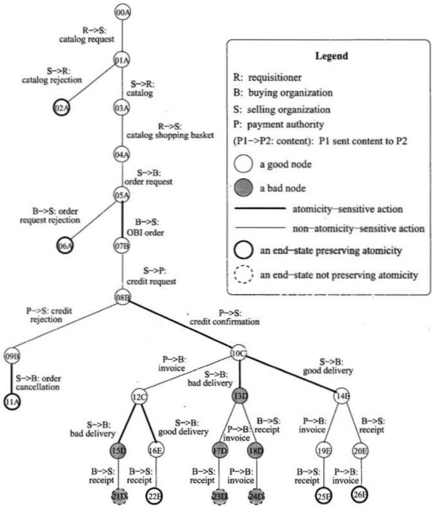


Figure 2.2: OBI Protocol Tree (Simplified Version)

A protocol tree is a pictorial representation of a protocol. For an EC protocol tree, there are following properties:

- Each path starting from the root is related to transaction executions of the protocol. It is a class of protocol executions that follow the same pattern. A path from the root to a leaf is a complete path, or a complete execution.
- Although all predictable executions should be represented as paths, not all "predictable" message sequence combinations are meaningful. For example, generally speaking, a message of goods should never precede a message of order because goods should not have been sent without a previous order. Hence, a path with goods preceding order should generally not be included in the protocol tree.
- The nodes of the tree are stages of protocol executions. Each node has a **content** representing the transaction state at that stage. Each node has a unique ID number followed by a letter denoting the corresponding content of the node. Two nodes have the same content if and only if they have the same letter inside.
- The arcs and paths are various kinds of state transitions. Each arc is labeled with the message that produces the state transition. A message can have an either **good** or **bad** property. A good message has the attributes consistent with protocol requirements. For instance, in Figure 2.2, "a good delivery" refers to a message of goods that is delivered in sufficient quality and quantity as required by the corresponding purchase order. A message is **atomicity-sensitive** if the

beginning and ending nodes of its corresponding arc have different contents. Atomicity-sensitive messages are usually those messages related to order, money and goods. Only atomicity-sensitive messages affect transaction atomicity.

- For each complete path that terminates at a state preserving atomicity, every node within the path is called a **good node**. Nodes other than good nodes are called **bad nodes**. A good node may lead to a state preserving atomicity, while a bad node cannot lead to any state preserving atomicity.

2.5 Supports for Dispute Handling

A dispute handling process consists of three typical steps: dispute initiation, investigation and decision making.

2.5.1 Dispute Initiation

A player in the EC system may initiate a dispute when the player is not satisfied with the execution result of some transaction. This player is termed **initiator** of the dispute. The dispute initiator contacts the arbiter and submits to it a **complaint** describing how he has been treated unfairly by other players, and a **request** that he thinks will recover his loss. The request is usually a set of statements each of which contains an action that the initiator wishes to be taken. In addition to the complaint and the request, the initiator should submit as well the identification number of the transaction for which the dispute is raised so that the arbiter is able to make further investigations.

2.5.2 Investigation

During the investigation step the arbiter tries to construct the current transaction state. Let Q →s R be a state transition where Q is the initial state and R the terminating state of s. Suppose a dispute arises when the transaction is in state R. The arbiter can construct state R if he knows sequence s by interacting with players and acquiring necessary information about s. According to the protocol structure, s is composed of various messages exchanged between players, among which only atomicity-sensitive messages affect the state of R. Therefore, if the arbiter is able to learn exactly what atomicity-sensitive messages have been exchanged so far, current state R can be constructed based on the protocol tree. Then, it will be clear whether or not the dispute initiator's complaint is true and whether or not the request is honorable.

However, constructing a complete current state is not always possible because some players may not be trustworthy. When the arbiter collects information about exchanged messages from players, it is possible that some players may not be willing to tell the truth. To deal with this dilemma, the notion of **benefit set** is proposed in [24]. A benefit set for a player is a set of propositions (or statements). Each proposition states either occurrence or non-occurrence of some atomicity-sensitive message. Presumably, a player would not refuse to prove propositions in his/her benefit set because such proofs do not compromise his/her best interests. That is, for an atomicity-sensitive message M, if M is a good message (pointing to a good node in the protocol tree), its sender's benefit set should contain a proposition stating occurrence of M and other players' benefit sets should contain propositions stating non-occurrence of M. Otherwise, if M is a bad message (pointing to a bad node from a good node), its sender's benefit set should state non-occurrence of M and other players' benefit sets should state occurrence of M. Algorithm 2.1 used for constructing benefit sets is proposed in [24].

Algorithm 2.1: Constructing Benefit Sets

Let Benefit(X) be the benefit set for player X and Sender(M) be the sender of message M. Based on the protocol tree, execute the following:

- For each atomicity-sensitive message M whose corresponding arc stops at a good node, include into Benefit(Sender(M)) a proposition claiming Sender(M) sent M, and into Benefit(J) a proposition claiming Sender(M) did not send M, where J is a different player from Sender(M).
- For each remaining atomicity-sensitive message M, include into Benefit(Sender(M)) a proposition claiming Sender(M) did not send M, and into Benefit(J) a proposition claiming Sender(M) sent M, where J is a different player from Sender(M).

Table 2.1: Players' Benefit Sets of OBI Protocol

buying_organization:
B1: buying_organization sent OBI_order to selling_organization
B2: payment_authority did not send credit_confirmation to selling_organization
B3: selling_organization did not send order_cancellation to buying_organization
B4: selling_organization did not send good delivery to buying_organization
B5: selling_organization sent bad delivery to buying_organization
selling_organization:
S1: buying_organization did not send OBI_order to selling_organization
S2: payment_authority did not send credit_confirmation to selling_organization
S3: selling_organization sent order_cancellation to buying_organization
S4: selling_organization sent good delivery to buying_organization
S5: selling_organization did not send bad delivery to buying_organization
payment_authority:
P1: buying_organization did not send OBI_order to selling_organization
P2: payment_authority sent credit_confirmation to selling_organization
P3: selling_organization did not send order_cancellation to buying_organization
P4: selling_organization did not send good delivery to buying_organization
P5: selling_organization sent bad delivery to buying_organization

Table 2.1 lists players' benefit sets of OBI protocol. It is easy to verify that it can be constructed by applying algorithm 2.1 based on OBI protocol tree in Figure 2.2.

The significance of benefit sets lies in theorem 2.1, which is proposed by the authors in [24].

Theorem 2.1: *Let R~N~S be a complete path in a protocol tree. Then an atomicity-sensitive message is in segment R~N if and only if its occurrence is claimed by a proposition in some benefit set that is true at node N. An atomicity-sensitive message is in segment N~S if and only if its non-occurrence is claimed by a proposition in some benefit set that is true at node N and its occurrence is claimed by another proposition in some benefit set that is true at node S.*

This theorem guarantees that the ·transaction state at any node of the protocol execution can be completely described by benefit set propositions that are true at that node. Hence, should a dispute arise at some node, we are able to try to construct the transaction state for that node by inspecting truth values of all benefit set propositions. This can facilitate our dispute handling because the complete transaction state is the first thing we need to know when applying atomicity as correctness criterion.

For instance, in Figure 2.2, the transaction state at node 15D is described by true benefit set propositions B1, B3, B4, B5, P2, P3, P4, and P5. That is, buying organization has sent OBI order to selling organization, payment authority has sent credit confirmation to selling organization, selling organization did not send order cancellation to buying organization, and selling organization has sent bad delivery to buying organization.

2.5.3 Decision Making

It is in this phase that the decision is made on whether the dispute initiator's request should be honored or not. Making the decision is based on the current transaction state that can be possibly constructed, and purchase atomicity is applied as the correctness criterion.

If complete knowledge on the current transaction state can be attained after players have presented their information, a decision can be generated by applying some algorithm which is intended to install atomicity. Otherwise, human involvement is necessary. Human experts analyze the dispute case by considering whatever other factors which might be helpful with regard to a resolution. Those factors may include, for instance, any additional documents players can provide, etc.

From above, it is appropriate to implement the arbiter as a two tier structure which includes both software arbiter and human arbiter. Software arbiter is a piece of computer software that generates algorithmic solutions, while human arbiter is composed of domain experts and provides human judgments whenever necessary.

Following is the algorithm introduced in [24] that should be executed by software arbiter in order to generate a decision.

Algorithm 2.2: Decision Generation

```
if truth values of all benefit set propositions that take value of true can be obtained
    construct the complete transaction state;
    if the state preserves atomicity
        if the dispute initiator can prove the complaint
            ask human arbiter to consider the reasonableness of the request;
        else // the initiator cannot prove the complaint
            no action is taken;
        end if
    else // the state does not preserve atomicity
        if the initiator's request reinstalls atomicity
            accept the request;[1]
        else // the request does not reinstall atomicity
            if the initiator can prove the complaint
                ask human arbiter to consider the reasonableness of the request;
            else // the initiator cannot prove the complaint
                no action is taken;
            end if
        end if
    end if
else // only partial state can be constructed
    ask human arbiter for judgment;
end if
```

1. Because atomicity is our ultimate correctness criterion, any request that preserves or reinstalls atomicity should be justifiable. Therefore, there is no need to care about whether the initiator can prove the complaint or not.

2.6 Using Rules to Prove Propositions

From the decision generation algorithm 2.2, proving benefit set propositions is a critical step since it is the basis of transaction state construction. However, due to some possible reasons, such as lost evidence and inherent deficiency of a protocol, it is not always feasible for players to prove benefit set propositions directly by showing corresponding evidence. Players may need to do the inference. This is done by means of **rules** [24]. A rule is an implication p→q where both p and q are propositions claiming either occurrence or non-occurrence of some messages. The rule p→q is used to prove q in case q is true by proving p. That is, p→q empowers us to show that q is true by showing that p is true. This is desirable if proving p is easier than proving q. By applying rules, players are more likely to be able to prove benefit set propositions.

2.7 Related Work

EC transactions have been studied extensively recently. A comprehensive review of research issues and challenges in EC field is given in [13]. Many EC protocols are proposed with varying levels of security guarantee. The NetBill protocol [6] ensures fair exchange for the sale of low-priced digital goods by adopting a trusted third party. Secure Electronic Transactions (SET) is a commercially developed standard aimed at EC transactions conducted by three players: customer, merchant, and credit card company [15]. The iKP family of secure electronic payment protocols is proposed in [5]. The iKP protocols implement credit card based transactions between the customer and the merchant while using existing financial network for clearing and authorization. The protocols can also be extended to apply to other payment models, such as debit cards and electronic checks. In [22], a set of EC protocols for micropayments is designed with the main goal to reduce the charging cost by choosing a suitable security model, a charging model, and cryptographic algorithms.

Atomicity is a property that has been thoroughly investigated in database transactions during the last two decades [9,16]. Studying atomicity in EC transactions is introduced in [26,27]. The author discusses the role of atomicity in EC and proposes three types of EC atomicity, namely, money atomicity, goods atomicity, and certified delivery. As an extension, another type of atomicity, distributed purchase atomicity, is proposed in

[21], where the authors address the lack of support for full atomicity in EC payment and apply transactional process management to realize an EC Payment Coordinator. In [28], the author analyzes in details the need of a transaction model, the corresponding transactional mechanism, and its usefulness for EC.

Yet the topic of handling EC disputes is outside the scope of the above works, although they usually do specify what evidence should be stored for a fair resolution for possible disputes. The work in [4] proposes a framework for dispute handling, which has been applied in the European SEMPER project [14]. The authors design a claim language for disputes independent of any specific payment system. They also describe a framework for dispute handling where a dispute protocol is developed. However, the work does not address the correctness criterion for resolutions in a unified manner. Consequently, it remains unclear how to adapt different payment systems to the framework. Also, although the work provides mechanisms for proving statements based on evidence, it presents few strategies to deal with cases where evidence is either lost or withheld.

Another work which addresses dispute handling issues is presented in [19]. An optimistic fair exchange protocol with automated dispute resolution is proposed. The protocol ensures true fair exchange and does not require manual dispute resolution in case of unfair behavior by any party. Nonetheless, the protocol is useful only for the exchange of digital items because the basis of the protocol is a mathematical theory for cross validation of messages. As a result, the protocol has only limited applications.

American Arbitration Association proposes a B2B EC Dispute Management Protocol [3]. The protocol provides pre-defined rules and procedures for handling disputes of different categories. It also employs an online web system to facilitate the dispute handling procedure. Through the web system, EC players are able to communicate with the arbitration system efficiently, e.g., filing a dispute case online. However, the dispute resolution indeed relies on human mediation and/or arbitration. The dispute management protocol does not involve automated dispute handling, e.g., decision generation by computer systems.

Chapter 3

An Architecture for EC Disputes

As mentioned in Chapter 1, our architecture uses a client-server model. Figure 3.1 shows a pictorial view of this architecture.

3.1 Client Applications

There are two kinds of client applications in our architecture. One kind is used for the interaction between EC players and the arbiter server. Running such a client application, each player is able to communicate with the arbiter server, e.g., to initiate a dispute.

The other kind of client applications serves as an interface through which the arbiter server components can be properly managed. For instance, initialization of protocol tree server can be done by field experts through corresponding client tools for protocol tree specification.

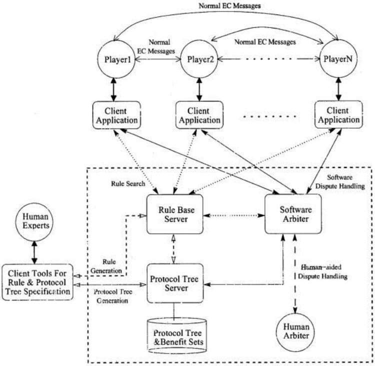


Figure 3.1: The Architecture of an EC Dispute Arbiter

3.2 Software Arbiter and Human Arbiter

Software arbiter and human arbiter are key components of the architecture. Software arbiter is the overall coordinator within the architecture. It interacts with various players, retrieves information from other components, etc. Also, software arbiter is responsible for execution of arbitration algorithms.

Due to the inherent complexity related to dispute handling (e.g., consider the comparable process conducted in a real court room by human judges), it is not always possible to solve disputes by software arbiter alone. Hence, necessary human assistance for arbitration is appropriate, especially in those cases where no clear-cut information is available. Thus, when software arbiter fails to reach a resolution by itself, it turns the case to human arbiter. Human arbiter actually provides knowledge and judgments from domain experts.

3.3 Protocol Tree Server

Protocol tree server deals with services and storage related to the protocol tree structure. The information provided by the protocol tree is essential in our architecture. For example, EC messages, transaction states, and benefit set propositions are all based on the protocol tree. Protocol tree server is equipped with a back-end database that stores necessary persistent data, such as tree structure, players' benefit sets, etc.

An important function of protocol tree server is protocol tree generation. Human experts may accomplish this through client applications for protocol tree specification. To facilitate the understanding of how a protocol tree can be generated, we first introduce a general model for EC protocols, and then present a protocol tree building algorithm.

3.3.1 A Model for EC Protocols

Because an EC protocol is a collection of rules that stipulate how EC transactions are executed, the protocol can be represented by listing all transaction executions that follow the protocol definition. As mentioned before, in our model, we further require that when representing a protocol those executions which do not follow the protocol definition should also be listed as long as they are predictable. (Because disputes are usually related to those "bad" executions, we need such information to handle disputes.) Therefore, in our protocol model, an EC protocol is a list of all predictable transaction executions.

On the other hand, as introduced previously, an EC transaction can be regarded as a sequence of message exchanging. Since a protocol prescribes several options for the ways a transaction under the protocol can proceed, an EC protocol can be viewed as a colloection of lists of message exchanging sequences, each of which represents a unique transaction execution.

In order to represent sequences of message exchanging, we need to define the general form of a message. A message here refers to any kind of information/item that is passed from one player to another. Hence, a message can be either a pure text flow or some goods with a physical shape. Following is the message definition adopted in our model:

(message_ID, sending player, receiving player, content, property)

Message_ID is a unique number identifying a particular message. Sending player refers to the sender of the message and receiving player refers to the receiver of the message. Content denotes what the message is, e.g., order or goods, etc. Property is a description used to represent some attribute values of the message. For instance, when the message content is some goods sent by a merchant to a customer, the goods may have an either "good" or "bad" property, which indicates whether or not both quality and quantity of the goods are consistent with what is stated in the order/contract.

Based on the above message definition, we are able to list all messages exchanged in the EC system. Then, it is feasible to design an algorithm that builds the protocol tree automatically.

3.3.2 A Protocol Tree Building Algorithm

In this section, we design an algorithm for building protocol trees. The algorithm constructs the protocol tree from a list of complete message sequences. Thus, before the algorithm can possibly work, some preparations need to be done to represent an EC protocol as a list of message sequences.

Three Preparation Steps

- Step 1: Defining Basic Sets

  Before composing all predictable messages, first we need to find out what possible values are for each value field of any message. According to our message definition, there should be following **basic sets**, each of which corresponds to some particular value fields of a message.

Player set: all possible players within the EC system. For any message, both the sending player field and the receiving player field take some values from the player set. For instance, in B2C model, a typical player set may be: merchant, customer, and bank.

Content set: all information flows/items that are exchanged between different players. All values for the content field of a message come from this content set.

Property set: all attribute values used to describe various message contents. Typical values of the property set are "good" and "bad", etc.

- Step 2: Composing Messages Exchanged in the Protocol

  Because all the basic sets have been defined in step 1, all possible values for any field of a message are known. Thus, composing all messages is a simple task of filling blank fields for messages.

- Step 3: Constructing the collection of all the possible message sequences

  Because each message sequence corresponds to a category of transaction executions, essentially, this step lists all predictable complete executions of the EC protocol, which imply the structure of the protocol tree.

It is clear that the above three steps demand thorough knowledge of the related EC protocol. Together, they serve to express an EC protocol as a list of complete message sequences. Therefore, all preparation steps should be accomplished by protocol experts.

The Algorithm

Algorithm 3.1 is the tree building algorithm that takes a list of complete message sequences as input and produces a protocol tree structure as output.

Algorithm 3.1: Protocol Tree Building

```
Seq_Left := set of all complete message sequences;
// Seq_Left is the list of remaining unprocessed sequences
create Root_Node; // This is the root node of the protocol tree

// Generate the first complete path in the protocol tree
Parent_Node := Root_Node; // Parent_Node is a variable of tree nodes
Seq := a sequence from Seq_Left with N messages;
// Seq is a variable of message sequences
for(i:=0; i<N; i:=i+1)
    create a new Child_Node as a child of the Parent_Node;
    // Child_Node is another variable of tree nodes
    label Arc(Parent_Node, Child_Node) with message_i of Seq;
    // Arc is a variable of tree arcs
```

```
  // Arc(A,B) represents the arc from node A to node B
  Parent_Node := Child_Node;
end for
remove Seq from Seq_Left;


// Generate other complete paths in the protocol tree
while Seq_Left is not empty
  Parent_Node := Root_Node;
  Seq := a sequence from Seq_Left with N messages;
  for(i:=0; i<N; i:=i+1)
    Found := FALSE;  // Found is used to denote whether or not message_i
                     // corresponds to an outgoing arc of Parent_Node
    for each Child_Node that is a child of Parent_Node
      if Arc(Parent_Node, Child_Node) is labeled with message_i
        Parent_Node := Child_Node;
        Found := TRUE;
        Break;  // It has been found that message_i corresponds to one outgoing
                // arc of Parent_Node, so there is no need to search the
                // remaining outgoing arcs of Parent_Node
      end if
    end for
```

```
        if(Found==FALSE)
            break; // Here, message_i does not correspond to any outgoing arc of
                    // Parent_Node, hence there is no need to search matching arcs for
                    // message_i, message_(i+1), ... , message_N
        end if
    end for
    for( ; i<N; i:=i+1)
        create a new Child_Node as a child of the Parent_Node;
        label Arc(Parent_Node, Child_Node) with message_i;
        Parent_Node := Child_Node;
    end for
    remove Seq from Seq_Left;
end while

// The remaining codes finish the algorithm
for each node in the tree
    assign a unique ID and a content label to the node;
end for
for each message in the tree
    if the two end-nodes of the message have different contents
        label the message as atomicity-sensitive;
    end if
end for
for each leaf node in the tree
    if the node preserves atomicity
        label it as an end-state preserving atomicity;
    else
        label it as an end-state not preserving atomicity;
    end if
end for
for each complete path in the tree
    if the path terminates at an end-state preserving atomicity
        for each node in the path
            label it as a good node;
        end for
    end if
end for
for each node in the tree
    if the node is not a good node
        label it as a bad node;
    end if
end for
```

An Example — Building OBI Protocol Tree

In this subsection, we build the OBI protocol tree as an example to show how the tree building algorithm works.

To apply the tree building algorithm, we first need to finish preparation steps. According to the descriptions of simplified OBI protocol introduced in section 2.3, there should be the following basic sets. Player set: requisitioner, buying organization, selling organization, payment authority. Content set: catalog request, catalog rejection, catalog, catalog shopping basket, order request, order request rejection, OBI order, credit request, credit rejection, credit confirmation, order cancellation, invoice, delivery of goods, receipt. Property set: good, bad. (For simplicity, we do not consider the digital signature and certificate scheme in OBI.)

Based on the defined basic sets, we can compose all the messages exchanged in the protocol. Please note that attention should be paid to those messages with particular properties, e.g., good and bad. Here are the composed messages: (requisitioner is referred to as R, buying organization as B, selling organization as S, payment authority as P, and empty value as Null.)

(1, R, S, catalog request, Null)

(2, S, R, catalog rejection, Null)

(3, S, R, catalog, Null)

(4, R, S, catalog shopping basket, Null)

(5, S, B, order request, Null)

(6, B, S, order request rejection, Null)

(7, B, S, OBI order, Null)

(8, S, P, credit request, Null)

(9, P, S, credit rejection, Null)

(10, S, B, order cancellation, Null)

(11, P, S, credit confirmation, Null)

(12, P, B, invoice, Null)

(13, S, B, delivery, bad)

(14, S, B, delivery, good)

(15, B, S, receipt, Null)

The last preparation step is to list all predictable complete message sequences, which requires a careful analysis of the protocol. It should be well understood that not all sequence combinations of messages are meaningful because in a message exchanging protocol there are usually some temporal orders imposed on messages. That is, in a specific protocol, some messages should always precede others. For example, in OBI, message 14 should always precede message 15 because the receipt from buying organization to selling organization can only be sent after selling organization has delivered products to buying organization. Therefore, even though the message sequences where message 15 precedes message 14 are predictable sequences, they are actually invalid options and hence should not be listed. In general, to check whether a message sequence is invalid or not, we need to check the order of every pair of messages within the message sequence. As long as there is a pair of messages whose order is not allowed (or is impossible) in the given protocol, the message sequence is viewed as invalid.

Here are complete message sequences we have recognized for the OBI protocol:

(1, 2)

(1, 3, 4, 5, 6)

(1, 3, 4, 5, 7, 8, 9, 10)

(1, 3, 4, 5, 7, 8, 11, 12, 13, 15)

(1, 3, 4, 5, 7, 8, 11, 12, 14, 15)

(1, 3, 4, 5, 7, 8, 11, 13, 12, 15)

(1, 3, 4, 5, 7, 8, 11, 13, 15, 12)

(1, 3, 4, 5, 7, 8, 11, 14, 12, 15)

(1, 3, 4, 5, 7, 8, 11, 14, 15, 12)

Finally, taking the above collection of complete message sequences as input, the tree building algorithm 3.1 produces the protocol tree illustrated in Figure 2.2 as output.

3.4 Rule Base Server

We have introduced previously that players may use rules to prove benefit set propositions. By applying rules, players are more likely to be able to prove propositions.

Rule base server is the architecture component dealing with issues related to the management of rules. Due to its vital importance in ensuring the correct functioning of the dispute handling architecture, we include the discussions on its strategies and functionalities in the next Chapter.

3.5 Dispute Handling in the Architecture

Our architecture handles EC disputes in the following way: the dispute initiator raises a dispute by contacting software arbiter. The initiator needs to submit the corresponding transaction ID, his/her complaint and request via architecture client application. Software arbiter then retrieves benefit sets from protocol tree server and sends each benefit set to its corresponding player. Also, software arbiter sends the transaction ID to players for their reference. Each player is asked to prove all propositions that he/she believes to be true in his/her benefit set, with regard to the transaction identified by the received ID. Players may prove propositions either directly or by applying some rules. If a player chooses to use rules, he/she has to search rule base server to find proper rules. After proofs are done, players send their proving results back to software arbiter. Software arbiter then executes the decision generation algorithm, either resolves the dispute itself, or turns the case over to the human arbiter. During this process, software arbiter may need assistance from human arbiter. Finally, software arbiter generates a dispute resolution and sends it to all players. Please note, before the architecture can work appropriately, some initializations have to be completed, such as building protocol tree, constructing benefit sets, and generating rules. These can be accomplished by human experts through corresponding client tools.

In the following, we present two scenarios based on OBI protocol to illustrate how disputes are handled in our architecture. The corresponding protocol tree is in Figure 2.2 (see page 14) and players' benefit sets are in Table 2.1 (see page 19).

Scenario 1:

Consider a case in which buying organization has placed an OBI order and paid to selling organization. However, the goods buying organization has received from selling organization are non-functional. When the transaction reaches node 15D, buying organization initiates a dispute. The complaint is 'selling organization made a bad delivery to us' and the request is 'selling organization provide an exchange for good goods'. After software arbiter receives these, it retrieves the benefit sets for all players, namely, buying organization, selling organization, and payment authority. Then, software arbiter sends benefit sets to their corresponding players and asks for proofs.

There are eight true propositions at node 15D: B1, B3, B4, B5, P2, P3, P4, P5. For buying organization, it proves B1 by showing a copy of the electronically signed OBI order. Also, it proves B5, and therefore B4, by presenting that the goods received from selling organization are indeed bad (not in accordance with the original OBI order). B3 is hard to prove directly since it is a proposition claiming non-occurrence of messages.

Hence, buying organization tries to apply rules. It contacts rule base server and finds the rule B5→B3. This is a reasonable rule because if B5 is true then B3 must be true according to OBI protocol tree (we discuss more on rules in next chapter). Hence, buying organization selects the rule B5→B3 and uses B5 to prove B3. Since B5 has been proved, buying organization proves B3 as well. For selling organization, none of its benefit set propositions is true. Hence, it cannot prove anything. For payment authority, P3, P4, P5 are the same as B3, B4, B5, respectively. Because B3, B4, and B5 are proved by buying organization, payment authority needs only to prove P2. P2 is not easy to prove directly, so payment authority selects the rule B5→P2 from rule base server. This is a good rule because B5 cannot be true unless P2 is true, based on the protocol tree. Because B5 is proved to be true by buying organization, payment authority hence proves P2 through B5→P2.

After collecting all proof results from players, software arbiter executes algorithm 2.2 to make a decision. Software arbiter finds no problem to construct the complete transaction state at node 15D because all true benefit set propositions have been proved. The state does not preserve atomicity, nevertheless buying organization's request reinstalls it. According to the algorithm, line 11 is reached. Hence, software arbiter honors buying organization's request.

Scenario 2:

In this example, suppose after buying organization has sent an OBI order and paid for the order the dishonest selling organization does not deliver any goods. That is, the transaction proceeds to node 10C and selling organization does not deliver any goods. Consequently, buying organization initiates a dispute. The complaint is 'we made payment but did not receive goods' and the request is 'selling organization make a good delivery'. In this case, true propositions in players' benefit sets are B1, B3, B4, S5, P2, P3, P4. Yet buying organization cannot prove B4. Neither can selling organization prove S5. Hence, after software arbiter collects proof results from players, it is unable to construct the complete current transaction state since not all true propositions have been actually proved. Therefore, according to line 21 of algorithm 2.2, software arbiter has to hand over the case to human arbiter for.judgment.

Chapter 4

Rule Processing: A Framework and Methodologies

As discussed in section 2.6, players may need to use rules to prove propositions because it is not always feasible for them to prove benefit set propositions directly by showing corresponding evidence. In this chapter, we take an in-depth look at the issues involved in the rule processing, such as the notion of weak rules, how to search for rules, how to measure their reliabilities, and how to cope with the possible inconsistencies among the weak rules.

4.1 Weak Rules

Traditionally, a rule is associated with a value to be respected by followers. In our context, this value is the truth. As briefly mentioned in Section 2.6, we can use the rule $p \rightarrow q$ to prove q by proving p. But a precondition is that p being true *always* implies q being true. A rule of this kind is called a *strong rule.* However, as realized by the authors in [23], in many cases strong rules are not obtainable. Thus they introduced the concept of *weak rules.* A weak rule does not have to be always true, and therefore, its convincing power is limited. (We will use the terms 'reliability' and 'convincing power' interchangeably in the subsequent discussions.) In cases where strong rules are not available, weak rules are the only feasible alternatives. What we would like to have are weak rules with high enough reliabilities so that when they are used for proof purposes the results generated are still acceptable to all the players (mostly importantly, though, the arbiter).

Look at the following example in OBI. Suppose selling organization wants to prove the proposition q 'selling organization sent good delivery to buying organization' by applying the rule p→q where p reads 'payment authority sent credit confirmation to selling organization'. This rule is a weak rule, meaning that it is not fully reliable. This is because the fact that p is true does not necessarily imply that q is true, according to the protocol tree in Figure 4.1. Nevertheless, the rule is practically meaningful because sometimes selling organization may not be able to prove good delivery by showing the receipt acquired from buying organization since a bad buying organization may withhold the receipt deliberately. Also good delivery of products is usually the hot spot for disputes.

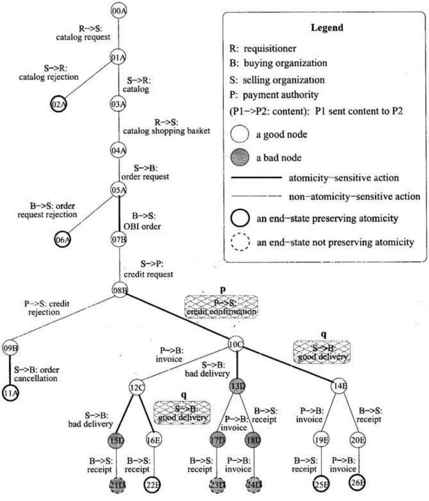


Figure 4.1: An Example of Weak Rules

On the other hand, it may be highly probable that selling organization can get credit confirmation from payment authority without much difficulty in that the latter is a third party independent from buying organization. Moreover, because the credit confirmation is the piece of evidence that selling organization would receive from payment authority, selling organization should normally keep it in records. That is, selling organization should have no difficulty in presenting the credit confirmation whenever it is needed, e.g., in the case when applying the rule stated above to prove that good delivery has been sent.

Therefore, the rule 'payment authority sent credit confirmation to selling organization' → 'selling organization sent good delivery to buying organization' is of practical importance to players because it can facilitate players to prove propositions, although it is only a weak rule.

4.2 Searching Heuristically for Rules

Because rules involve using one proposition to prove another and propositions are related to messages, the generation of rules requires finding relationships and relative positioning between messages. This suggests the need to search for rules on the EC protocol tree since this tree provides the required information on messages. The heuristic method presented below extends the one introduced in [23]. It generates both strong and weak rules. In addition, the rules generated can contain either positive or negative propositions.

(we call a proposition **positive** if it claims occurrence of a message, and a proposition **negative** if it claims non-occurrence of a message.)

In the following we will use p to denote a positive proposition and $\neg$ p to denote a negative proposition that claims the negation of p.

Algorithm 4.1: Searching for Rules

Input: a protocol tree T;

Output: a set of candidate rules, R();

```
for each positive proposition p that claims occurrence of a message m
    1. $R1 \leftarrow R2 \leftarrow R3 \leftarrow \varnothing$, and mark every complete path that includes m;
    2. for each message n such that n appears in every marked complete path and n is ahead of m
           $R1 \leftarrow ( R1 \cup \{p \rightarrow q\} )$ where q is the proposition claiming occurrence of n;
       end for
    3. for each message k such that all paths that contain k should also contain m and m is ahead of k in at least one of those paths
           $R2 \leftarrow ( R2 \cup \{p \rightarrow q\} )$ where q is the proposition claiming occurrence of k;
       end for
    4. for each message j in S which is the set of messages that are not in any marked path
```

```
            R3 ← ( R3 ∪ {p → ¬ q} ) where ¬ q is the proposition claiming non-
            occurrence of j;
        end for
    5.  for each rule p → q within R1
            R1 ← ( R1 ∪ {¬ q → ¬ p} );
        end for
    6.  for each rule p → q within R2
            R2 ← ( R2 ∪ {¬ q → ¬ p} );
        end for
    7.  for each rule p → ¬ q within R3
            R3 ← ( R3 ∪ {q → ¬ p} );
        end for
    8.  R() ← ( R1 ∪ R2 ∪ R3 );
end for
```

In the algorithm, steps 2, 3, and 4 find rules based on the relative positioning between messages in the protocol tree. Then, steps 5, 6, and 7 add contrapositives of existing rules into the candidate sets.

The motivation for step 2 is as follows. If message n appears in every path containing message m and n is ahead of m in the path then the occurrence of m must imply the occurrence of n because the transaction execution cannot reach m without first passing n. The rules in this set R1 are hence fully reliable. Look at the two messages n and m in Figure 4.2 where n is "B->S: OBI order" and m is "P->S: credit confirmation". In this case, n appears in every path that contains m and n is always ahead of m, thus, the occurrence of m should imply the occurrence of n. That is, the rule p → q should be generated where p claims occurrence of m and q claims occurrence of n.

On the other hand, although step 3 adopts a principle similar to that of step 2, not all rules generated in this case are fully reliable. This is because in step 3 the occurrence of message m is used to imply the occurrence of message k even if m is ahead of k, which is not guaranteed to be true. In Figure 4.2, for instance, let message m be "B->S: OBI order" and message k "S->B: good delivery". Then, every path containing k also contains m and m is ahead of k in at least one path. So, the rule p → q where p claims occurrence of m and q claims occurrence of k is generated in step 3. It is easy to observe that this rule is not fully reliable because the occurrence of "OBI order" does not guarantee the occurrence of following up message "good delivery". Nevertheless, we still need this set R2 of rules generated in step 3 because we try to give players more choices of possible rules. Clearly, rules that are not fully reliable need special treatment so that errors resulted from applications of these rules can be avoided as much as possible. We discuss more on this issue in the following sections.

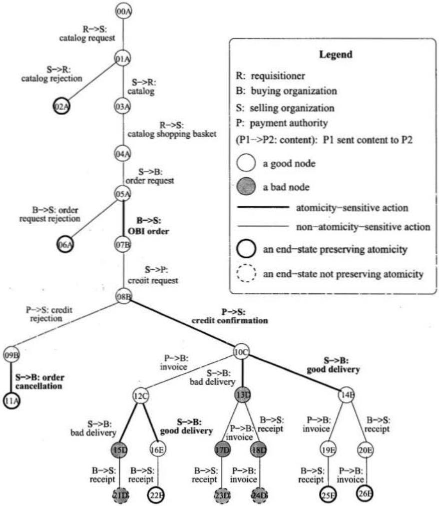


Figure 4.2: Rule Search Examples

Step 4 deals with propositions claiming non-occurrences of messages. If two messages are not in the same path, occurrence of one must imply non-occurrence of the other because the transaction execution can follow only one path at a time. For example, in Figure 4.2, the message m "S->B: order cancellation" and the message j "P->S: credit confirmation" are not in the same path. Therefore, m and j cannot occur simultaneously. The rule $p \rightarrow \neg q$ should be generated during step 4 where p claims occurrence of m and $\neg q$ claims non-occurrence of j. Rules in R3 generated in step 4 are all fully reliable.

4.3 Rule Validity Weight and Its Calculation

According to algorithm 4.1, normally there should be more than one rule in the candidate set and these rules may have different reliability degrees. Not all rules are equally reliable. Some rules are fully reliable, meaning that for a rule $p \rightarrow q$ in case p is true q must be true as well. Then, it has no problem when applying these rules. A fully reliable rule is termed a **valid** rule. On the other hand, some rules are not fully reliable. That is, the rule $p \rightarrow q$ does not guarantee that q is always true when p is true. A rule that is not fully reliable is termed a **weak** rule. Weak rules have different *reliability degrees*. To evaluate and compare reliability degrees of rules, we introduce the **validity weight** of a rule.

The validity weight of a rule $p \rightarrow q$ reflects the reliability degree of proving q by proving p. That is, the weight tells the probability for q to be true in case p has already been proved true. We therefore use the conditional probability $P(q|p)$ for the weight of rule p→q. For example, if p→q is a valid rule, it has a full weight of value one because P(q|p) equals to 1 in this case.

Because P(q|p) = P(pq) / P(p), we are able to calculate P(q|p) by first calculating P(pq) and P(p). As is known, each complete path in the protocol tree represents a possible route for the transaction execution. And for a specific transaction, it must follow only one path. Hence, different paths in the protocol tree are mutually exclusive for a particular transaction execution. This enables us to calculate both P(pq) and P(p) by applying Bayes' formula. That is, we determine P(pq) and P(p) by conditioning upon whether or not the transaction has followed a specific path.

To calculate P(pq), we use the following formula (suppose there are N complete paths in the protocol tree):

$$P(pq) = P(\ pq \mid path\ 1\ ) * P(\ path1\ ) + P(\ pq \mid path\ 2\ ) * P(\ path2\ ) + \ldots$$
$$+ P(\ pq \mid pathN\ ) * P(\ pathN\ )$$

P(pq | pathPT) is the probability for the events of p and q under the condition that the transaction has followed the path PT. To calculate P(pq | pathPT), we need to find out TrueNode(pq, pathPT) and AllNode(pathPT). TrueNode(pq, pathPT) is the number of nodes on path PT where both p and q are true. AllNode(pathPT) is the number of all nodes on path PT. Then, we have the following:

$$P(\ pq \mid pathPT\ ) = TrueNode(\ pq, pathPT\ ) / AllNode(\ pathPT\ )$$

P(pathPT) is the probability for the execution to follow path PT. At this point, we suppose for each edge in the protocol tree there is a corresponding probability, termed *edge probability*, which states how probable the transaction should follow this edge. Then the path probability P(pathPT) is the product of probabilities of all edges in path PT. In the next subsection, we present more details on edge probability and how it is determined.

Similar to the calculation of P(pq), we have the following formula to calculate P(p):

$$P(p) = P(\, p \mid \text{path } 1\,) * P(\, \text{path1}\,) + P(\, p \mid \text{path } 2\,) * P(\, \text{path2}\,) + \ldots$$

$$+ P(\, p \mid \text{path } N\,) * P(\, \text{pathN}\,)$$

$$\text{where } P(\, p \mid \text{pathPT}\,) = \text{TrueNode}(\, p, \text{pathPT}\,) \,/\, \text{AllNode}(\, \text{pathPT}\,)$$

Based on the above, we design algorithm 4.2 that is used for calculating validity weights.

Algorithm 4.2: Calculating Rule Weights

Input: a rule $p \rightarrow q$;

Output: the validity weight $P(\, p \rightarrow q\,)$;

```
P( pq ) = 0;
P( p ) = 0;
for each path N in the protocol tree, execute the following steps:
  calculate the path probability by multiplying all probabilities of edges in path N, i.e.,
            $P(\, pathN\,) = P(\, edge1\,) * P(\, edge2\,) * \ldots * P(\, edgeM\,)$
  AllNode( path N ) = count all nodes in path N;
  TrueNode( p, path N ) = count all nodes in path N that are within the sub-path of p;
```

```
 // On a complete path containing p, the sub-path of p is the portion from the node
 // immediately after p to the leaf node of the complete path
 if p is ahead of q in path N
   TrueNode( pq, path N ) = count all nodes in path N that are within the sub-path of q;
 else // q is ahead of p in path N
   TrueNode( pq, path N ) = count all nodes in path N that are within the sub-path of p;
 end if
 $P( p ) = P( p ) + P( pathN ) * ( TrueNode(p, path N)/AllNode(path N) );$
 $P( pq ) = P( pq ) + P( pathN ) * ( TrueNode(pq, path N)/AllNode(path N) );$
end for
$P( q|p ) = P( pq ) / P( p );$
$P( p \rightarrow q ) = P( q|p );$
```

In Section 4.1 above, we have given an example for weak rules. We now apply algorithm 4.2 to calculate the validity weight for that rule.

According to algorithm 4.2, first we need to consider path probabilities. From the OBI protocol tree (see Figure 4.3), it can be found that there are 9 paths in total:

Path1: 00A...01A...02A

Path2: 00A...03A...06A

Path3: 00A...08B...11A

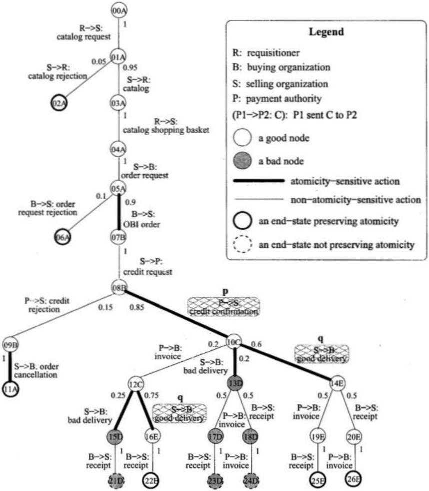


Figure 4.3: An Example of Rule Weight Calculation

Path4: 00A...08B...10C...12C...21D

Path5: 00A...08B...10C...12C...22E

Path6: 00A...08B...10C...13D...23D

Path7: 00A...08B...10C...13D...24D

Path8: 00A...08B...10C...14E...25E

Path9: 00A...08B...10C...14E...26E

In Figure 4.3, each edge is associated with a probability. That is the probability under which the transaction would follow the corresponding edge. With the probabilities provided in the figure, we are able to calculate path probabilities according to the following formula: (Refer to Section 4.4 for detail on how to calculate the related edge probabilities.)

P(pathN) = P(edge1) * P(edge2) * ... * P(edgeM)

P(path1) = 1*0.05 = 0.05

P(path2) = 1*0.95*1*1*0.1 = 0.095

P(path3) = 1*0.95*1*1*0.9*1*0.15*1 = 0.12825

P(path4) = 1*0.95*1*1*0.9*1*0.85*0.2*0.25*1 = 0.0363375

P(path5) = 1*0.95*1*1*0.9*1*0.85*0.2*0.75*1 = 0.1090125

P(path6) = 1*0.95*1*1*0.9*1*0.85*0.2*0.5*1 = 0.072675

P(path7) = 1*0.95*1*1*0.9*1*0.85*0.2*0.5*1 = 0.072675

P(path8) = 1*0.95*1*1*0.9*1*0.85*0.6*0.5*1 = 0.218025

P(path9) = 1*0.95*1*1*0.9*1*0.85*0.6*0.5*1 = 0.218025

Next, find out for each pathN the values of TrueNode(pq, pathN) and AllNode(pathN):

TrueNode(pq, path1) = 0 AllNode(path1) = 3

P(pq | path1) = TrueNode(pq, path1) / AllNode(path1) = 0

TrueNode(pq, path2) = 0 AllNode(path2) = 6

P(pq | path2) = TrueNode(pq, path2) / AllNode(path2) = 0

TrueNode(pq, path3) = 0 AllNode(path3) = 9

P(pq | path3) = TrueNode(pq, path3) / AllNode(path3) = 0

TrueNode(pq, path4) = 0 AllNode(path4) = 11

P(pq | path4) = TrueNode(pq, path4) / AllNode(path4) = 0

TrueNode(pq, path5) = 2 AllNode(path5) = 11

P(pq | path5) = TrueNode(pq, path5) / AllNode(path5) = 2/11

TrueNode(pq, path6) = 0 AllNode(path6) = 11

P(pq | path6) = TrueNode(pq, path6) / AllNode(path6) = 0

TrueNode(pq, path7) = 0 AllNode(path7) = 11

P(pq | path7) = TrueNode(pq, path7) / AllNode(path7) = 0

TrueNode(pq, path8) = 3 AllNode(path8) = 11

P(pq | path8) = TrueNode(pq, path8) / AllNode(path8) = 3/11

TrueNode(pq, path9) = 3 AllNode(path9) = 11

P(pq | path9) = TrueNode(pq, path9) / AllNode(path9) = 3/11

Then, we can get the value of P(pq):

P(pq) = P(pq | path 1) * P(path1) + P(pq | path 2) * P(path2) + …

+ P(pq | pathN) * P(pathN)

= 0*0.05 + 0*0.095 + 0*0.12825 + 0*0.0363375

+ (2/11)*0.1090125 + 0*0.072675 + 0*0.072675 + (3/11)*0.218025

+ (3/11)*0.218025

= 0.138743

Similarly, find for each pathN the values of TrueNode(p, pathN) and AllNode(pathN):

TrueNode(p, path1) = 0 AllNode(path1) = 3

P(p | path1) = TrueNode(p, path1) / AllNode(path1) = 0

TrueNode(p, path2) = 0 AllNode(path2) = 6

P(p | path2) = TrueNode(p, path2) / AllNode(path2) = 0

TrueNode(p, path3) = 0 AllNode(path3) = 9

P(p | path3) = TrueNode(p, path3) / AllNode(path3) = 0

TrueNode(p, path4) = 4 AllNode(path4) = 11

P(p | path4) = TrueNode(p, path4) / AllNode(path4) = 4/11

TrueNode(p, path5) = 4 AllNode(path5) = 11

P(p | path5) = TrueNode(p, path5) / AllNode(path5) = 4/11

TrueNode(p, path6) = 4 AllNode(path6) = 11

P(p | path6) = TrueNode(p, path6) / AllNode(path6) = 4/11

TrueNode(p, path7) = 4 AllNode(path7) = 11

$P(\ p \mid path7\ ) = TrueNode(p, path7) / AllNode(path7) = 4/11$

$TrueNode(p, path8) = 4 \quad AllNode(path8) = 11$

$P(\ p \mid path8\ ) = TrueNode(p, path8) / AllNode(path8) = 4/11$

$TrueNode(p, path9) = 4 \quad AllNode(path9) = 11$

$P(\ p \mid path9\ ) = TrueNode(p, path9) / AllNode(path9) = 4/11$

And, we can get the value of P(p):

$$\begin{aligned}
P(p) &= P(\ p \mid path\ 1\ ) * P(\ path1\ ) + P(\ p \mid path\ 2\ ) * P(\ path2\ ) + \ldots \\
&\quad + P(\ p \mid pathN\ ) * P(\ pathN\ ) \\
&= 0*0.05 + 0*0.095 + 0*0.12825 \\
&\quad + (4/11)*0.0363375 + (4/11)*0.1090125 + (4/11)*0.072675 \\
&\quad + (4/11)*0.072675 + (4/11)*0.218025 + (4/11)*0.218025 \\
&= 0.264273
\end{aligned}$$

Finally, we are able to calculate the validity weight for the rule p→q:

$$P(\ p \to q\ ) = P(\ q|p\ ) = P(pq) / P(p) = 0.138743 / 0.264273 = 0.525$$

If the validity weight of a rule equals one, this rule is a full-weight rule. There are some general principles that can be used to identify full-weight rules easily. One principle is concerned with the temporal sequence order between various messages claimed in rules. Because a transaction execution is actually a series of ordered message exchanges, the occurrence/non-occurrence of a particular message could possibly be implied by the occurrence/non-occurrence of some other messages.

Message 1 **precedes** message 2 in protocol P, if with regard to the whole protocol tree of P, message 2 can only be reached via paths including message 1 where message 1 is ahead of message 2.

Message Sequence Maxim: *if message 1 precedes message 2 in protocol P, then occurrence of message 2 implies occurrence of message 1 and non-occurrence of message 1 implies non-occurrence of message 2.*

In the heuristic search algorithm, R1 is the set that contains all rule candidates generated according to message sequence maxim. Hence, rule candidates in R1 should be of full validity weight because they are totally reliable. However, candidates in R2 do not comply with the message sequence maxim and thus they do not have full weights.

The message sequence maxim complies with algorithm 4.2 in the sense that for any rule p→q, if the message mentioned in p implies the message mentioned in q, then the algorithm will tell that P(pq) equals to P(p), which means that P(q|p) equals one (the full weight).

Another principle about full-weight rules deals with **contradicting messages**. If two messages, message 1 and message 2, are never on the same path, they are called contradicting messages. Contradicting messages cannot both be true in the same transaction execution because they are not on the same path, and any practical protocol execution can follow only one path at a time. If a rule has the form of p→q where p claims occurrence of message 1 and q claims non-occurrence of message 2, i.e., occurrence of message 1 → non-occurrence of message 2, then the rule should have full weight. The rule candidate set R3 in heuristic search algorithm is composed of rules generated according to the principle of contradicting messages. Hence, rule candidates in R3 should be of full validity weight.

The principle of contradicting messages also complies with algorithm 4.2. For the rule p→q, where p states occurrence of message 1, q states non-occurrence of message 2, and message 1 contradicts with message 2, P(pq) must be equal to P(p) because in the protocol tree wherever p is true q must be true as well. Hence, P(q|p) should equal one (the full weight)

4.4 Determining Edge Probability

In the algorithm for calculating validity rule weight, there are two kinds of probabilities: **path probability** and **edge probability**. A path probability refers to the probability under which the transaction follows this particular path. Because each path is actually composed of many edges, to calculate path probability, we need to know the probability under which the transaction would follow each edge on the path. The probability associated with each edge is called edge probability that represents the probability for the transaction to follow the corresponding edge.

For edge probability, we have the following observation. The sum of edge probabilities of all outgoing edges for any node in the protocol tree equals one. This is because all outgoing edges of a node represent all the possibilities that the transaction may chose to execute immediately after this node. Since all outgoing edges represent all execution possibilities which is the whole space, the sum of probabilities of all outgoing edges equals one which is the probability for the whole space.

Then, a question arises: how to determine the edge probability associated with each edge in the protocol tree? Is there any well-formatted formula that can decide how probable it is for the transaction to choose a particular edge out of all outgoing edges of a node?

Before we try to answer the above questions, first look at the following instance based on Figure 4.1 Node 05A contains one incoming edge and two outgoing edges. The incoming edge is 'S->B: order request' that means selling organization sent order request to buying organization. So, node 05A corresponds to the state the transaction reached after selling organization sent the OBI order request to buying organization. After this state, there are two possibilities the transaction may choose: the order request would be either rejected or accepted. Should the order request be rejected, the message 'buying organization sent order request rejection to selling organization' would take place. This corresponds to one of the two outgoing edges for node 05A. On the other hand, should the order request be accepted, buying organization would send the OBI order which is based on the order request to selling organization. That is, the message 'buying organization sent OBI order to selling organization' would take place in this case and it corresponds to the other outgoing edge of node 05A. Therefore, to determine the probabilities for the two outgoing edges, we need to know the probabilities for rejection and acceptance of the OBI order request.

However, it is not straightforward to determine in advance the probability of either rejection or acceptance. Take the rejection for example. The OBI order request could be rejected by buying organization due to many reasons. The requisitioner might have ordered something that he/she was not authorized to order. Buying organization might not have enough funding to make the purchase. A manager might decide that some items were not necessary. For a particular execution, it is hard, if not impossible, to determine in advance whether some rejecting reasons would occur or not, and what combination of rejecting reasons it would be.

Therefore, we choose to determine edge probability by experience. That is, we determine edge probability by analyzing historical data. (It is generally agreed that past experiences are good indications of future. For instance, when admitting new students, university officials usually try to predict prospective students' future performances based on their past academic records.)

The method that we use to analyze experience data is described in the following. Suppose we consider edge E that is an outgoing edge for node N in the protocol tree. Let No be the number of outgoing messages associated with E that have occurred in the past and Ni the number of incoming messages associated with N that have occurred in the past.

Suppose past messages can be reasonably recorded and stored into a message inventory, i.e., numbers of past messages are traceable. Consequently, both No and Ni are available. Then we have:

edge probability of E = No / Ni

That is, we calculate edge probability of E by investigating the proportion that No takes away from Ni.

Let us return to the previous example. For node 05A in Figure 4.1, the incoming message is 'S->B: order request' that represents the order request sent from selling organization to buying organization. If buying organization keeps a record of the number of all order requests that it has received during the past, then the value of Ni for node 05A is available. We assume it is 300. The outgoing message 'B->S: order request rejection' represents the order request rejection sent from buying organization to selling organization. If selling organization keeps a record of the number of all order request rejections that it has received in the past, then the value of No for edge 05A—06A is available as well. We assume it is 36. Finally, we would be able to determine edge probability for edge 05A—06A:

edge probability of edge 05A—06A = No / Ni = 36 / 300 = 0.12

That is, according to records, the order request rejection rate is 12 out of 100. This is a good indication of how probable a future order request might be rejected. Hence, we can set the probability of edge 05A—06A to 0.12.

4.5 Determining the Acceptance Criterion

With the validity weight, we are able to evaluate how reliable a rule is. According to the algorithm of rule weight assignment, different rules may have different weights. Clearly, we should not allow players to apply those rules whose weights are very low because the low weight of a rule indicates that the rule is not so reliable. Then, a practical issue arises: how to determine the **acceptance criterion** for rule weights. The acceptance criterion is a threshold value such that all rules whose weights are equal or above the criterion are viewed as acceptable and all rules whose weights are below the criterion are viewed as unacceptable. Players are allowed to apply only acceptable rules when trying to prove propositions via applications of rules.

We determine the acceptance criterion for rule weights based mainly on practical requirements and experiences. If the requirements are in favor of a large rule set containing many acceptable rules available for players to use, the acceptance criterion can be set to a smaller value so that more rules can pass the criterion. Note that the smaller the criterion is, the lower reliability a passing rule may have. Otherwise, if a small rule set containing highly reliable rules is expected, the acceptance criterion should be set to a large value. An extreme case is that the acceptance criterion is set to the value of one, which is equal to the full weight. In this case, only rules with full weights (fully reliable rules) are allowed to be applied by players. On the other hand, practical experiences should also be given considerations. When determining the acceptance criterion, we pay attention to the acceptable rules in a practical sense. After we set a criterion and acquire a set of acceptable rules, we may choose some rules out of the set and analyze how acceptable the rules indeed are in reality. If we find some rules that are acceptable based on the criterion are actually unacceptable according to our practical experiences, e.g., contradicting to some common sense, then we have to increase the acceptance criterion in order to fix the problem.

Finally, the value of acceptance criterion must not be less than 0.5. The argument is as follows. As introduced in section 4.3, the weight of rule $p \rightarrow q$ is actually the conditional probability $P(q|p)$. In order for the rule $p \rightarrow q$ to be valid, there must be $P(q|p)>=P(\neg q|p)$. Also, base on the probability theory, we have $P(q|p)+P(\neg q|p)=1$. Then,

$$1=P(q|p)+P(\neg q|p)<=P(q|p)+P(q|p)=2P(q|p)$$

Therefore, $1<=2P(q|p)$. That is, $P(q|p)>=0.5$, which indicates that the weight of any valid rule must not be less than 0.5. Consequently, the acceptance criterion for rules must not be less than 0.5.

4.6 An Example Application of Weak Rules

The introduction of weak rules does not change the decision generation algorithm, but the applications of weak rules do affect players' abilities of proving propositions. Now, with choices of weak rules, players are more likely to be able to prove benefit set propositions. The following example shows how weak rules can be applied in the dispute handling to assist players. The example is based on OBI protocol. For convenient reference, we list again the OBI protocol tree and benefit sets in Figure 4.4 and in Table 4.1, respectively.

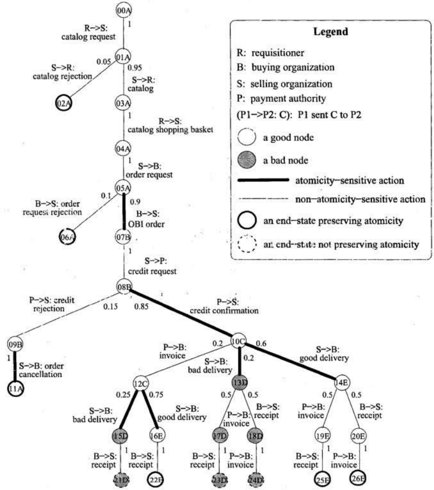


Figure 4.4: OBI Protocol Tree (Simplified Version)

Table 4.1: Players' Benefit Sets of OBI Protocol

buying_organization:

B1: buying_organization sent OBI_order to selling_organization

B2: payment_authority did not send credit_confirmation to selling_organization

B3: selling_organization did not send order_cancellation to buying_organization

B4: selling_organization did not send good delivery to buying_organization

B5: selling_organization sent bad delivery to buying_organization

selling_organization:

S1: buying_organization did not send OBI_order to selling_organization

S2: payment_authority did not send credit_confirmation to selling_organization

S3: selling_organization sent order_cancellation to buying_organization

S4: selling_organization sent good delivery to buying_organization

S5: selling_organization did not send bad delivery to buying_organization

payment_authority:

P1: buying_organization did not send OBI_order to selling_organization

P2: payment_authority sent credit_confirmation to selling_organization

P3: selling_organization did not send order_cancellation to buying_organization

P4: selling_organization did not send good delivery to buying_organization

P5: selling_organization sent bad delivery to buying_organization

Suppose the transaction execution reaches node 16E, when the good delivery of products has been sent from selling organization to buying organization. At this point, the execution is fine and the transaction state preserves atomicity. However, the dishonest buying organization tries to gain some extra benefits from selling organization. Hence, buying organization does not return the receipt back to selling organization and initiates a dispute by contacting the arbiter system. The submitted complaint is 'selling organization sent a bad delivery of products to us' and the request is 'selling organization provide us some compensations'. After software arbiter receives these, it sends benefit sets to the corresponding players and asks for proofs.

For selling organization, it cannot prove any proposition in its benefit set directly. Thus, selling organization has to explore using rules. At the current state, the message credit confirmation has been sent from payment authority to selling organization. Consequently, selling organization keeps credit confirmation and should have no problem to show it. Therefore, selling organization contacts rule base server for some rules that can make use of credit confirmation. Selling organization finds the rule 'payment authority sent credit confirmation to selling organization' → 'selling organization sent good delivery to buying organization' particularly helpful. This rule, as discussed before, is a weak rule and its validity weight is 0.525. Suppose the acceptance criterion for weak rules used in the architecture is 0.51. Then, the rule is acceptable and allowed to be applied by players. Since selling organization has no problem to prove the proposition 'payment authority sent credit confirmation to selling organization' by showing the evidence, i.e., credit confirmation, selling organization is able to apply successfully the rule to prove benefit set proposition S4. Once S4 is proved, selling organization can further prove S5 by applying another rule S4 → S5, which is a full-weight rule.

For payment authority, its benefit set proposition P2, i.e., 'payment authority sent credit confirmation to selling organization', has been proved by selling organization. Then payment authority can use the rule P2 → P3 to prove P3 with no difficulty because the rule has a full weight, meaning it is fully reliable.

For buying organization, it is unable to prove B1 directly. But buying organization contacts rule base server and finds the full-weight rule 'payment authority sent credit confirmation to selling organization' → B1. Since the proposition 'payment authority sent credit confirmation to selling organization' has been proved, buying organization can apply the rule successfully. Hence, B1 is proved. There is no need for buying organization to prove B3 because B3 is the same as P3, which has been proved. Nevertheless, buying organization cannot prove B4 and B5 either directly or by applying rules. For instance, the rule 'payment authority sent credit confirmation to selling organization' → B5 has a partial weight of merely 0.175 based on the rule weight calculation algorithm, which is below the acceptance criterion.

From above, the proved benefit set propositions are B1, B3, S4, S5, P2, and P3. After the proof results are sent back from players, software arbiter executes algorithm 2.2 for decision generation. Based on the proof results, software arbiter is able to construct the complete transaction state which is represented by letter E in the protocol tree. Since this state preserves atomicity, software arbiter asks the dispute initiator, i.e., buying organization, to prove the complaint, according to line 4 of algorithm 2.2. Because the complaint claims a bad delivery and what buying organization has received is actually a good delivery, buying organization must be unable to prove the complaint. Then, according to line 7 of algorithm 2.2, no action is taken. That is, buying organization's request is refused.

The above example illustrates the significance of weak rules. In the general case, if weak rules are not used then this dispute is very hard to handle. This is because buying organization does not return the receipt and selling organization therefore has trouble to prove the benefit set proposition 'selling organization sent good delivery to buying organization'. A critical step during the dispute handling is the application of the weak rule 'payment authority sent credit confirmation to selling organization' → 'selling organization sent good delivery to buying organization', which helps selling organization out of the dilemma.

4.7 Inconsistency of Rules

As discussed previously, players can use rules to prove propositions. Suppose a player wants to apply the rule p → q to prove proposition q. If he can prove p directly, then he is done, otherwise, he can try to apply another rule k → p to prove p. Suppose he can prove k directly. Then p is proved. Consequently, players now can apply the rule p → q successfully to prove q. In this case, two successful applications of rules k → p and p → q enable players to prove proposition q by proving proposition k directly. The applications of k → p and p → q can be conveniently represented as k → p → q. If the player still cannot prove k directly, we can repeat the above inference. (Note: a player can prove a proposition directly implies that the proposition is true.) In the following we formalize the idea in the general case.

Definition 4.1: *Let T be a protocol tree and N be a node in T. A sequence $p1 \rightarrow p2 \rightarrow p3 \rightarrow \ldots \rightarrow p(n-1) \rightarrow pn$ is called a* ***proof sequence at node*** *N if:*

1. For all i, $1 <= i <= n-1$, rule $pi \rightarrow p(i+1)$ exists, and its validity weight is above the acceptance criterion;

2. p1 is true at node N.

Definition 4.2: *A set of rules is* ***inconsistent*** *if there exist two sequences of rules in the set, $S1 = p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$ and $S2 = r1 \rightarrow r2 \rightarrow r3 \ldots \rightarrow r$, and a node N in the protocol tree, such that*

1. Both S1 and S2 are proof sequences at node N;

2. p and r are conflicting, meaning they either are negation forms of each other, or claim the occurrences of contradicting messages.

If a set of rules is inconsistent, then conflicting propositions may be produced. That is, by applying rules in a rule set that is inconsistent, conflicting results, e.g., positive and negative forms of the same proposition, can be implied simultaneously.

More specifically, because different players may apply different proof sequences of rules to prove their propositions, if the rule base is inconsistent it is possible that two players will end up with proving conflicting propositions, e.g., p and ¬p, by applying different proof sequences. This may render it impossible for the arbiter to reach a decision because p and ¬p cannot be both true simultaneously in reality.

4.7.1 Consistency Theorem

For a set of full-weight rules, we have the following theorem.

Theorem 4.1: *A set of rules containing only full-weight rules generated by the heuristic search algorithm is consistent.*

Proof:

According to the definition of inconsistent rule sets, if we can show that there do not exist two sequences with conflicting endings, this set of rules cannot be inconsistent. Let p1 → p2 → p3 ... → p and r1 → r2 → r3 ... → r be arbitrary two sequences. We prove the following two cases:

Case 1: it is impossible that p and r are negation forms of each other.

Case 2: it is impossible that p claims occurrence of message m1, r claims occurrence of message m2, yet m1 and m2 are contradicting messages.

Proof for Case 1:

Assume the contrary. Let p and r be negation forms of each other. Thus we have $r = \neg p$. So, we need to show $p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$ and $r1 \rightarrow r2 \rightarrow r3 \ldots \rightarrow \neg p$ cannot exist simultaneously.

From the heuristic search method, a positive proposition can only be implied by another positive proposition, while a negative proposition can be implied by another proposition that is either positive or negative.

Therefore, we need to show that the following two sets of sequences cannot exist for a given set of full-weight rules (for convenience, we use a single character to represent a positive proposition and use symbol ¬ followed by a single character to represent a negative proposition):

Set 1: some negative proposition ¬k1 is used to prove ¬p by applying a sequence of rules where only negative propositions are involved.

$$p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$$

$$\neg k1 \rightarrow \neg k2 \rightarrow \neg k3 \ldots \rightarrow \neg p$$

Set 2: some positive proposition m1 is used to prove ¬p by applying a sequence of rules where both positive and negative propositions are involved.

$p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$

$m1 \rightarrow m2 \rightarrow m3 \ldots \rightarrow mk \rightarrow \neg n1 \rightarrow \neg n2 \ldots \rightarrow \neg p$

First of all, it is true that if $p1 \rightarrow p2$ and $p2 \rightarrow p3$ then $p1 \rightarrow p3$. The full-weight rule $p1 \rightarrow p2$ means that each path containing p1 should also contain p2 and p2 is ahead of p1. A similarly result exists for $p2 \rightarrow p3$. Then it can be derived that each path containing p1 should also contain p3 and p3 should be ahead of p1, i.e., $p1 \rightarrow p3$.

Next, we can show that $p1 \rightarrow p2$ iff $\neg p2 \rightarrow \neg p1$. In fact, this is guaranteed by the message sequence maxim.

Suppose sequences in set 1 were possible. Using the above two facts, we have:

$p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p \Rightarrow p1 \rightarrow p$

$\neg k1 \rightarrow \neg k2 \rightarrow \neg k3 \ldots \rightarrow \neg p \Rightarrow p \rightarrow \ldots k3 \rightarrow k2 \rightarrow k1 \Rightarrow p \rightarrow k1$

Next,

$p1 \rightarrow p \rightarrow k1$

$\Rightarrow p1 \rightarrow k1$

$\Rightarrow \neg k1 \rightarrow \neg p1$

That is, each path containing p1 should also contain k1. Hence, p1 and $\neg k1$ cannot be true simultaneously. Therefore, the following sequences cannot exist simultaneously:

$p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$

$\neg k1 \rightarrow \neg k2 \rightarrow \neg k3 \ldots \rightarrow \neg p$

This contradicts our supposition. The supposition is hence incorrect and the sequences in set 1 cannot exist simultaneously.

Similarly, we can show that the sequences in set 2 cannot exist simultaneously, either. Again, assume the sequences existed:

$$m1 \rightarrow m2 \rightarrow m3 \ldots \rightarrow mk \rightarrow \neg n1 \rightarrow \neg n2 \ldots \rightarrow \neg p \Rightarrow \neg n1 \rightarrow \neg n2 \ldots \rightarrow \neg p$$

$$\neg n1 \rightarrow \neg n2 \ldots \rightarrow \neg p \Rightarrow p \rightarrow \ldots n2 \rightarrow n1 \Rightarrow p \rightarrow n1$$

And,

$$p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p \Rightarrow p1 \rightarrow p \Rightarrow p1 \rightarrow p \rightarrow n1 \Rightarrow p1 \rightarrow n1 \Rightarrow \neg n1 \rightarrow \neg p1$$

So,

$$m1 \rightarrow m2 \rightarrow m3 \ldots \rightarrow mk \rightarrow \neg n1 \rightarrow \neg n2 \ldots \rightarrow \neg p$$

$$\Rightarrow m1 \rightarrow m2 \rightarrow m3 \ldots \rightarrow mk \rightarrow \neg n1$$

$$\Rightarrow m1 \rightarrow mk \rightarrow \neg n1$$

$$\Rightarrow m1 \rightarrow mk \rightarrow \neg n1 \rightarrow \neg p1$$

The rule mk → ¬n1 means mk and n1 are not coexistent. They are not on the same path. The rule m1 → mk guarantees that each path containing m1 also contains mk, hence m1 and n1 are not on the same path. (Otherwise, suppose m1 and n1 were on the same path, then mk should also be on that path. This produces the conclusion that mk and n1 are on the same path, which is a contradiction to the rule mk → ¬n1.) There should be m1 → ¬n1. Therefore,

$$m1 \rightarrow mk \rightarrow \neg n1 \rightarrow \neg p1$$

$\Rightarrow m1 \rightarrow \neg n1 \rightarrow \neg p1$

Obviously, it is true that $m1 \rightarrow \neg n1 \Rightarrow n1 \rightarrow \neg m1$ because m1 and n1 are contradicting and occurrence of one implies non-occurrence of the other. Therefore,

$m1 \rightarrow \neg n1 \rightarrow \neg p1$

$\Rightarrow p1 \rightarrow n1 \rightarrow \neg m1$

$\Rightarrow p1 \rightarrow \neg m1$

$\Rightarrow m1 \rightarrow \neg p1$

That is, p1 and m1 are not coexistent, meaning that they are not on the same path. Hence, p1 and m1 cannot be true simultaneously for an execution. Therefore, the following sequences cannot exist simultaneously.

$p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$

$m1 \rightarrow m2 \rightarrow m3 \ldots \rightarrow mk \rightarrow \neg n1 \rightarrow \neg n2 \ldots \rightarrow \neg p$

This is, however, a contradiction to our assumption. Thus, the assumption is incorrect and the set 2 of sequences is impossible.

From above, we have shown that $p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$ and $r1 \rightarrow r2 \rightarrow r3 \ldots \rightarrow \neg p$ cannot exist simultaneously. Thus, case 1 is proved.

Proof for Case 2:

Suppose the sequence pair $p1 \rightarrow p2 \rightarrow p3 \ldots \rightarrow p$ and $r1 \rightarrow r2 \rightarrow r3 \ldots \rightarrow r$ existed where p and r claim contradicting messages. Because p and r claim contradicting messages, there must be such a rule $r \rightarrow \neg p$. Then, from the sequences $r1 \rightarrow r2 \rightarrow r3 \ldots \rightarrow r$ and $r \rightarrow$

¬p we get r1 → r2 → r3 ... → r → ¬p. Thus, two sequences p1 → p2 → p3 ... → p and r1 → r2 → r3 ... → ¬p exist simultaneously. However, this result contradicts the conclusion of case 1. The contradiction arises because of the false supposition. Hence, case 2 is proved.

We have proved both case 1 and case 2, therefore, the consistency theorem is proved.

4.7.2 An Example of Inconsistency Problem

Unfortunately, theorem 4.1 guarantees consistency only for rule sets composed of full-weight rules. If rules in a rule set are not all of full weight, the rule set may be inconsistent. This is because the properties and facts we have applied when proving theorem 4.1 are not available for rules that do not have full weights. For example, in the proof of theorem 4.1, arguments for sequences in Set 1 of Case 1 are invalid for partial-weight rules. That is, if the weight of p1 → k1 is not full, it is not safe to draw the conclusion that p1 and ¬k1 cannot be true simultaneously. The partial weight of p1 → k1 means that it is probable for some path containing p1 not to contain k1. Consequently, it is probable that both p1 and ¬k1 are true simultaneously. Then, sequences in Set 1 may exist simultaneously, which yield conflicting results, i.e., p and ¬p.

To make it clearer, we give an example of inconsistency problem. Consider the OBI protocol tree in Figure 4.5.

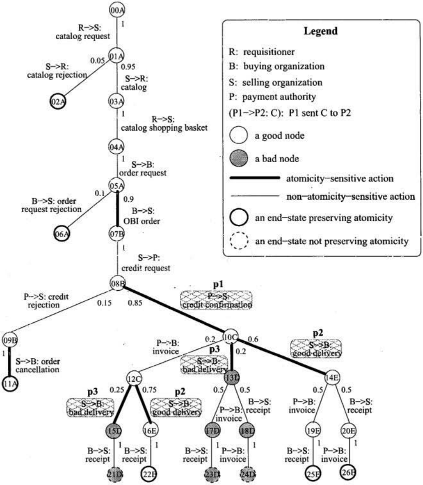


Figure 4.5: An Example of Inconsistent Rules

We are interested in three propositions here:

p1: payment authority sent credit confirmation to selling organization.

p2: selling organization sent good delivery to buying organization.

p3: selling organization sent bad delivery to buying organization.

And consider the rule set containing two rules: $\{ p1 \rightarrow p2 , p3 \rightarrow \neg p2 \}$. The rule $p3 \rightarrow \neg p2$ is a full-weight rule because p3 and p2 claim conflicting messages, i.e., good delivery and bad delivery. The rule $p1 \rightarrow p2$, nevertheless, has a partial weight only, because it does not comply with the message sequence maxim. As shown before, we calculate the weight of $p1 \rightarrow p2$ to be 0.525 by applying the algorithm of assigning rule weights. Suppose in the arbiter system the acceptance criterion for rule weights is 0.51, which means any rule having a weight no less than 0.51 is allowed to be applied by players. So, $p1 \rightarrow p2$ is an acceptable rule.

Because p2 is a proposition in selling organization's benefit set and $\neg p2$ is a proposition in buying organization's benefit set, it is possible that selling organization uses $p1 \rightarrow p2$ to prove p2 and buying organization uses $p3 \rightarrow \neg p2$ to prove $\neg p2$. If the transaction is currently at node 15D, then the message of credit confirmation should be held by selling organization and the bad delivery of products should be held by buying organization. Then, selling organization is able to apply successfully the rule $p1 \rightarrow p2$ to prove p2 and buying organization is able to apply successfully the rule $p3 \rightarrow \neg p2$ to prove $\neg p2$ as well. Consequently, a conflict arises because p2 and $\neg p2$ cannot be true simultaneously in reality. This renders it impossible for the arbiter to make a decision because the arbiter is confused about whether p2 is true or not.

Therefore, it is clear that the rule set { p1 → p2 , p3 → ¬p2 } is inconsistent and conflicts may be produced. The inconsistency comes from the fact that p1 → p2 is not a full-weight rule. As a result, there is a probability for some player, e.g., selling organization in the example, to use the rule successfully to prove p2 even when p2 is actually false which is proved by the full-weight rule p3 → ¬p2 in the above.

4.8 Handling Inconsistency Problem in the Rule Base

We have shown a rule set containing partial-weight rules may be inconsistent. Because rules stored on rule base server are generated according to the heuristic search algorithm, there are many partial-weight rules in the rule base. Therefore, the rule base is not guaranteed to be a consistent rule set. As a result, different rules applied by various players may produce conflicts. This problem is serious because conflicts, such as proving p and ¬p at the same time, make it impossible for the arbiter to reach a decision. Thus, a solution needs to be found to solve the inconsistency problem in the rule base.

4.8.1 The Algorithm for Handling Inconsistency Problem

In order to strengthen players' abilities to prove propositions, we have to include some partial-weight rules in the rule base. So, we are not able to avoid the inconsistency problem in advance by eliminating all inconsistent partial rules. Hence, conflicts may be produced. Conflicts, however, can be detected and removed. That is, if the arbiter is able to remove conflicts in a proper way once they are identified, no harm would be done to the arbiter system. Therefore, what we need is to design a strategy of identifying and removing conflicts.

Conflicts are results of applying partial-weight rules. Consider the previous example again. Two rules p1 → p2 and p3 → ¬p2 produce a conflict because p1 → p2 is a partial-weight rule. The event that p2 is false while p1 is true may happen due to the partial-weight of p1 → p2 that does not guarantee p2 to be true when p1 is true. When that event happens, p1 → p2 should not be applied. In fact, p3 → ¬p2 confirms the happening of that event since p3 → ¬p2 is a full-weight rule and it guarantees that p2 is false. Thus, the application of p3 → ¬p2 should prevent the simultaneous application of p1 → p2. That is, p3 → ¬p2 should remove p1 → p2. As a result, the conflict, i.e., p2 and ¬p2 are true simultaneously, can be eliminated.

A point worth noting here is concerned with "current true propositions". When applying rules, we estimate the true or false value of a proposition by the true value of another single proposition. For instance, when p1 → p2 is applied, we estimate the true or false value of p2 based on the true value of p1. However, it is clear that when the two rules p1 → p2 and p3 → ¬p2 are applied, true propositions in the current situation are p1 and p3, instead of the single proposition p1. Hence, it is more accurate to estimate either p2 or ¬p2 based on both propositions of p1 and p3. However, we observe that considering all true propositions will incur a high runtime overhead. Because complete current true propositions can only be known at runtime, the proposition estimation has to be done dynamically. That is, rule weight assignments have to be done when players are proving propositions because only at this time can complete current true propositions be learned. Also, rule weights have to be recalculated repeatedly and frequently because the set of current true propositions changes whenever a new proposition is proved. Thus we consider the antecedent only of a rule for the calculation of its conditional probability. This makes it possible for us to obtain the rule weights based only on the protocol tree during the initialization phase of rule base server.

Our approach to handle the inconsistency problem is termed **Wound & Remove**. The basic idea is that a rule with a higher weight should wound and remove another rule with a lower weight when the applications of two rules result in conflicts. This is because we believe a rule with a higher weight is more reliable.

We must generalize the Wound & Remove approach to deal with rule proof sequences because players may apply more than a single rule to prove propositions. As introduced in section 4.7, a proof sequence is a series of rules where the beginning proposition is used to prove the ending proposition through many applications of different rules, e.g., p1 → p2 → ... → pk. Particularly, a single rule p1 → p2 is the simplest form of a proof sequence. The weight of a proof sequence is the multiplication result of rule weights of all rules contained in the sequence. For instance, the weight of p1 → p2 → p3 equals to the multiplication result of p1→p2's rule weight and p2→p3's rule weight. The sequence weight reflects the probability of implication from the beginning proposition to the ending proposition. Then, the Wound & Remove approach can be generalized to deal with proof sequences: a proof sequence with a higher sequence weight should wound and remove another proof sequence with a lower sequence weight when the two proof sequences generate conflicting ending propositions.

Then, we are ready to present the algorithm of handling inconsistency problem in the rule base by adopting the Wound & Remove approach. Basically, we maintain for each player a record set where the following information is stored: all propositions the player has proven, the proof sequences applied to prove those propositions, and the corresponding proof sequence weights. If a proposition is proved directly then the applied proof sequence is deemed as empty and the corresponding proof sequence weight is set to value of 1, i.e., the full weight.

In order to handle inconsistencies, we think the Wound & Remove algorithm should perform the following functions:

1. When a player tries to submit a proof sequence in order to prove some proposition, the algorithm should check each proposition in the sequence against all propositions that have been proved and logged in the record sets associated with players. To check a proposition p, the algorithm should try to find whether or not there are some propositions conflicting to p in the record sets. If yes, the algorithm should determine the proposition with the highest sequence weight and remove all its conflicting

propositions. Note that all propositions in the submitted proof sequence should be checked because by submitting a proof sequence the player is attempting to prove all propositions in the sequence. For example, if the submitted sequence is p1 → p2 → p3 → p4 → p5, all propositions of p1, p2, p3, p4, and p5 should be checked although the player intends to prove the ending proposition p5.

2. If accepting all propositions in the proof sequence does not produce any conflict, the algorithm should accept the proof sequence and all its propositions by recording proper information in the player's record set. For the beginning proposition, the related proof sequence is recorded as empty and the sequence weight is value 1. For other propositions, record information in the following way. Find the sub-sequence for each proposition, and then record the proposition, the sub-sequence, and the sub-sequence weight. The sub-sequence for a proposition p, which is not the beginning proposition of the original sequence, is the portion from the beginning proposition to proposition p in the original sequence. For instance, if the original sequence is p1 → p2 → p3 → . . p4 → p5, then the sub-sequence for p3 is the portion p1 → p2 → p3.
3. If some propositions in the submitted proof sequence cause conflicts, do not perform any wounding until all propositions in the sequence have been checked. If the check results show that all propositions in the sequence have survived the wound and remove comparisons, accept the submitted sequence. Record all propositions, their corresponding sub-sequences, and the sub-sequence weights in the record set of the player who has submitted the sequence. Cancel all conflicting propositions by

removing the conflicting propositions, the corresponding proof sequences, and the sequence weights from the record sets. Cancel all affected proposition entries as well (those proposition entries whose corresponding proof sequences contain canceled propositions), by removing the affected proposition entries from the record sets. Note that it is proposition entries, instead of propositions, that should be removed in this case. This is because it is possible that one proposition may have multiple record set entries, each of which is associated with a unique proof sequence. Those entries, whose corresponding proof sequences do not contain any canceled proposition, are not affected by the proposition cancellations and hence should be preserved.

4. If some propositions or proposition entries are canceled, notify players whose record sets contain those propositions or entries, and request them to reprove those canceled items if necessary.
5. If the check results show that at least one proposition in the submitted proof sequence should be wounded and removed due to conflicts, the current submitted sequence should be canceled. Notify the player who has submitted the proof sequence that the sequence is not acceptable because of conflicts.

The following is the algorithm for handling inconsistencies. For each player we maintain a record set, where each record has three attributes: proposition name, the related proof sequence used to prove the proposition, and the proof sequence weight. ConflictFlag(p) is a flag denoting if there is some proposition conflicting to p. CompareFlag(p) is another flag denoting if proposition p has a higher sequence weight than its conflicting proposition.

Algorithm 4.3: The Wound and Remove Algorithm

```
receive the proof sequence p1 → p2 → ...... → pk submitted by player A;
// Check conflicts for every proposition in the sequence
for each proposition p in the proof sequence
    find all propositions in players' record sets that conflict to p;
    chose proposition q that has the highest sequence weight of all found propositions;
    if there is such a proposition q
        compare sequence weight of p with that of q;
        if sequence weight of p > sequence weight of q
            CompareFlag(p) = WIN;
        else
            CompareFlag(p) = LOSE;
        end if
        ConflictFlag(p) = TRUE;
    else
        ConflictFlag(p) = FALSE;
    end if
end for
// Based on the check results, perform either accepting or wounding
if ConflictFlag(p) is FALSE for every proposition p in the proof sequence
    accept the proof sequence;
    for each proposition p in the proof sequence
        if p is the beginning proposition of the proof sequence
            record the following information in the record set of player A:
                (p, ‘ ‘, 1);
        else
            record the following information in the record set of player A:
                (p, the sub-sequence for p, the sub-sequence weight);
        end if
    end for
else
    find all propositions in the proof sequence whose ConflictFlags are TRUE;
    if CompareFlag(p) is WIN for every proposition p that has been found
        accept the proof sequence;
        for each proposition pp in the proof sequence
            if pp is the beginning proposition of the proof sequence
        record the following information in the record set of player A:
            (pp, ‘ ‘, 1);
    else
        record the following information in the record set of player A:
            (pp, the sub-sequence for pp, the sub-sequence weight);
    end if
end for
for each proposition pw in the submitted proof sequence whose ConflictFlag is TRUE
    for each proposition pc in the record sets that conflicts to pw
        for each proposition entry of pc in the record sets
            cancel the proposition entry;
            notify the player whose record set contains the canceled entry;
            remove the following information from the record set:
                (pc, the proof sequence, the sequence weight);
        end for
        for each proposition entry whose proof sequence contains pc
            cancel the proposition entry;
            notify the player whose record set contains the canceled entry;
            remove the following information from the record set:
                (the canceled proposition, the proof sequence,
                 the sequence weight);
            end for
        end for
    end for
  else
    cancel the submitted proof sequence;
    notify player A that the submitted proof sequence is unacceptable;
  end if
end if
```

4.8.2 Some Examples of Applying the Wounding Algorithm

Here we present some scenarios to show how the wounding algorithm works to handle inconsistency problem in the rule base. Suppose there are three players in the system: player1, player2, and player3. And below is an inconsistent rule set:

$$\{p1\rightarrow^{0.7}p2;\ p2\rightarrow^{1}p3;\ p3\rightarrow^{1}p4;\ p5\rightarrow^{0.9}p3;\ p5\rightarrow^{1}p4;$$

$$p6\rightarrow^{1}\neg p2;\ p6\rightarrow^{1}\neg p3;\ p6\rightarrow^{1}\neg p7;\ \neg p7\rightarrow^{0.8}\neg p3;\}$$

According to the rule set, there is no conflict among p1, p2, p3, p4, p5 and p7. Proposition p6, nevertheless, conflicts to any of p2, p3 and p7. Table 4.2 can be used to store necessary information of the record sets required by the algorithm:

Table 4.2: Structure of Players' Record Sets

	Proposition Name	Proof Sequence	Sequence Weight
Player1			
Player2			
Player3			

Scenario 1:

Player1 submits the following proof sequence: p1$\rightarrow^{0.7}$p2$\rightarrow^{1}$p3$\rightarrow^{1}$p4. Because the current record sets are all empty, there is no conflict according to line3~17 of the algorithm. That is, the ConflictFlag(p) for each proposition of p1, p2, p3 and p4 is FALSE. Then, line19~30 of the algorithm are executed and the information is updated to Table 4.3.

Next, player3 tries to submit the proof sequence ¬p7$\rightarrow^{0.8}$¬p3. When the algorithm is executed to evaluate this sequence, one negation form of ¬p3, i.e., p3, is found in player1's record set. So ConflictFlag(¬p3) is TRUE. Also, that p3 is the proposition with the highest sequence weight of 0.7 in all record sets. Hence, the weight is compared with the currently submitted ¬p3's sequence weight, i.e., 0.8, according to line7. The comparison result is that the CompareFlag(¬p3) is assigned WIN, meaning proposition ¬p3 should wound and remove proposition p3 in the record sets.

Table 4.3: Recording Player1's Sequence — Scenario 1

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	‘ ‘	1
	p2	p1→p2	0.7
	p3	p1→p2-→p3	0.7
	p4	p1→p2→p3→p4	0.7
Player2			
Player3			

Table 4.4: Recording Player3's Sequence — Scenario 1

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	‘ ‘	1
	p2	p1→p2	0.7
	p3	p1→p2→p3	0.7
	p4	P1→p2→p3→p4	0.7
Player2			
Player3	¬p7	‘ ‘	1
	¬p3	¬p7→¬p3	0.8

Then, lines20~29 are ignored since ConflictFlag(¬p3) is TRUE. Instead, lines31~62 are executed. Because ¬p3 is the only proposition whose ConflictFlag is TRUE (¬p7 does not cause any conflict) and CompareFlag(¬p3) is WIN, the submitted sequence by player3 is accepted and lines33~42 are executed to record information. The updated record sets are listed in Table 4.4.

It is clear that there is a conflict in the table because both p3 and ¬p3 are there. However, this conflict can be removed by the execution of lines43~58. First of all, remove those proposition entries that are conflicting to the winner proposition ¬p3, based on lines45~50. Hence, all entries of p3 are removed from the record sets as shown in Table 4.5:

Table 4.5: Removing Entries of p3 — Scenario 1

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	' '	1
	p2	p1→p2	0.7
	p4	P1→p2→p3→p4	0.7
Player2			
Player3	¬p7	' '	1
	¬p3	¬p7→¬p3	0.8

Secondly, remove those entries that are affected by the cancellation of p3. That is, all the entries whose corresponding proof sequences contain p3 are also canceled based on lines51~56. In Table 4.5, entry of p4 is removed because its corresponding proof sequence contains p3, which has already been canceled during the last step. Therefore, Table 4.6 lists the final record sets:

Table 4.6: Final Record Sets of Scenario 1

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	‘ ‘	1
	p2	p1→p2	0.7
Player2			
Player3	¬p7	‘ ‘	1
	¬p3	¬p7→¬p3	0.8

Because all conflicting propositions and affected proposition entries have been wounded and removed, the remaining propositions are all acceptable.

Scenario 2:

Player1 submits the proof sequence: $p1\rightarrow^{0.7}p2\rightarrow^{1}p3\rightarrow^{1}p4$. As in scenario 1, the results are shown in Table 4.7:

Table 4.7: Recording Player1's Sequence — Scenario 2

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	' '	1
	p2	p1→p2	0.7
	p3	p1→p2→p3	0.7
	p4	p1→p2→p3→p4	0.7
Player2			
Player3			

Table 4.8: Recording Player2's Sequence — Scenario 2

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	' '	1
	p2	p1→p2	0.7
	p3	p1→p2→p3	0.7
	p4	p1→p2→p3→p4	0.7
Player2	p5	' '	1
	p3	p5→p3	0.9
Player3			

Different from scenario 1, however, player2 submits the sequence $p5 \rightarrow^{0.9} p3$ immediately after player1's submission and before player3 tries to submit the proof sequence $\neg p7 \rightarrow^{0.8} \neg p3$. Because neither of p5 and p3 conflicts to any proposition in the current record sets, player2's sequence does not cause any conflict. Hence, the sequence is accepted and propositions are added into player2's record set. Table 4.8 is the updated information.

Then, player3's sequence $\neg p7 \rightarrow^{0.8} \neg p3$ is submitted. Because the current record sets contain p3 already, ConflictFlag(¬p3) is TRUE. In fact, there are two entries of proposition p3 in the record sets and the one with the highest sequence weight is in player2's record set. Therefore, the sequence weight (0.9) of the entry of p3 in player2's record set is compared with that (0.8) of the currently submitted proposition ¬p3. The result is $0.9 > 0.8$ and hence proposition ¬p3 is wounded and removed. That is, CompareFlag(¬p3) is LOSE at line11. According to line59~62 of the algorithm, the currently submitted proof sequence by player3 is unacceptable and therefore canceled. Consequently, the arbiter sends a message to player3 notifying that the proof sequence is not acceptable.

Notice that the final result of this scenario is quite different from that of scenario 1. The difference results from the p3 entry in player2's record set, which has a higher sequence weight than the proposition ¬p3 submitted by player3.

Scenario 3:

First, player1 submits $p1 \xrightarrow{0.7} p2 \xrightarrow{1} p3 \xrightarrow{1} p4$. Since this is the first sequence, it is accepted. And Table 4.9 lists the resulting record sets:

Table 4.9: Recording Player1's Sequence — Scenario 3

	Proposition Name	Proof Sequence	Sequence Weight
Player1	P1	‘ ‘	1
	p2	p1→p2	0.7
	p3	p1→p2→p3	0.7
	p4	p1→p2→p3→p4	0.7
Player2			
Player3			

Next, player2 submits two sequences: $p5 \xrightarrow{0.9} p3$ and $p5 \xrightarrow{1} p4$. Because propositions in the submitted sequences, i.e., p3, p4, and p5, do not conflict to any proposition in the record sets, both sequences are accepted. As a result, the corresponding information is recorded in Table 4.10:

Table 4.10: Recording Player2's Sequence — Scenario 3

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	‘ ‘	1
	p2	p1→p2	0.7
	p3	p1→p2→p3	0.7
	p4	p1→p2→p3→p4	0.7
Player2	p5	‘ ‘	1
	p3	p5→p3	0.9
	p5	‘ ‘	1
	p4	p5→p4	1
Player3			

Finally, player3 tries to submit $p6\rightarrow^{1}\neg p3$. Proposition ¬p3 causes conflicts because there are already two entries of p3 in the log. Based on line4~16, CompareFlag(¬p3) is WIN in that the sequence weight for ¬p3 is 1 which is higher than any sequence weight of p3 in the log. In addition, p6 conflicts to both p2 and p3 in the record sets. CompareFlag(p6) is WIN because p6 is proved directly and has the full weight of 1. Therefore, the submitted sequence is accepted and added into Table 4.11:

Table 4.11: Recording Player3's Sequence — Scenario 3

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	' '	1
	p2	p1→p2	0.7
	p3	p1→p2→p3	0.7
	p4	p1→p2→p3→p4	0.7
Player2	p5	' '	1
	p3	p5→p3	0.9
	p5	' '	1
	p4	p5→p4	1
Player3	p6	' '	1
	¬p3	p6→¬p3	1

Then, all conflicting propositions and those affected proposition entries are removed to maintain the integrity of the log table. According to line45~50 of the algorithm, all conflicting proposition entries are removed. Particularly, the entry of p2 in player1's record set and the two entries of p3 in both record sets of player1 and player2 are removed, as shown in Table 4.12:

Table 4.12: Removing Entries of p2 and p3 — Scenario 3

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	‘ ‘	1
	p4	p1→p2→p3→p4	0.7
Player2	p5	‘ ‘	1
	p5	‘ ‘	1
	p4	p5→p4	1
Player3	p6	‘ ‘	1
	¬p3	p6→¬p3	1

Table 4.13: Final Record Sets of Scenario 3

	Proposition Name	Proof Sequence	Sequence Weight
Player1	p1	‘ ‘	1
Player2	p5	‘ ‘	1
	p5	‘ ‘	1
	p4	p5→p4	1
Player3	p6	‘ ‘	1
	¬p3	p6→¬p3	1

Also, all the entries whose corresponding proof sequences contain p2 and/or p3, i.e., affected proposition entries, are removed based on line51~56. So, the entry of p4 in player1's record set is removed since its corresponding proof sequence contains p2 and p3. The resulting Table 4.13 is therefore non-conflicting, i.e., there is no conflict left among proved propositions.

An interesting point of this example is that there is still an entry of p4 in player2's record set finally. This entry is not removed because it does not incur any problem, although another entry of p4 has been removed from player1's record set as an affected proposition entry, i.e., its proof sequence contains p2 and p3. Therefore, it is clear that some entries of an affected proposition may be left in the record sets. Entries of affected propositions are different from entries of conflicting propositions because the latter should be removed completely.

An OBI Scenario:

We have discussed in previous sections that the weak rule 'payment authority sent credit confirmation to selling organization' → 'selling organization sent good delivery to buying organization' is useful for players to prove propositions. However, since it is a weak rule, some player may misuse it. That is, some player may try to use the rule to prove the proposition 'selling organization sent good delivery to buying organization' even when the proposition is false. In this case, the misuse of this weak rule may cause conflicts. In this scenario, we give such an example and show how the wounding algorithm can remove the conflict and help the arbiter reach a correct decision. The corresponding OBI protocol tree is in Figure 4.4 and the benefit sets are in Table 4.1.

After the transaction reaches node 10C, selling organization makes a bad delivery to buying organization. Consequently, buying organization initiates a dispute. The complaint is 'selling organization made a bad delivery to us' and the request is 'selling organization provide an exchange for good goods'. Then, software arbiter requests players to prove their benefit set propositions.

First, selling organization submits the rule 'payment authority sent credit confirmation to selling organization'$\rightarrow^{0.525}$S4. Selling organization chooses this rule because it can prove 'payment authority sent credit confirmation to selling organization' by showing the credit confirmation it holds. This is a weak rule, but its weight 0.525 is above the acceptance criterion, which is supposed to be 0.51 in the system. So, the proof is accepted. Let p be 'payment authority sent credit confirmation to selling organization', then the corresponding information is added into Table 4.14. After this, selling organization submits another proof sequence $p\rightarrow^{0.525}S4\rightarrow^{1}S5$. The sequence has a weight of 0.525, which is acceptable, too. Hence, the proof is also accepted and the resulting record sets are listed in Table 4.14:

Table 4.14: Recording Selling Organization's Sequences

	Proposition Name	Proof Sequence	Sequence Weight
Selling Organization	p	‘ ‘	1
	S4	p→S4	0.525
	p	‘ ‘	1
	S4	p→S4	0.525
	S5	p→S4→S5	0.525
Buying Organization			
Payment Authority			

Next, buying organization proves B5 directly by showing the bad product received from selling organization. Once B5 is proved, buying organization further submits the following three full-weight rules: $B5\rightarrow^{1}B1$, $B5\rightarrow^{1}B3$, and $B5\rightarrow^{1}B4$. Consequently, the record sets are updated as shown in Table 4.15.

At this point, the arbiter finds there are some conflicts in the record sets. That is, B5 conflicts to both S4 and S5. The arbiter therefore executes the Wound & Remove algorithm to eliminate conflicts. From Table 4.15, B5 has a sequence weight of 1, while both S4 and S5 have the sequence weight of 0.525. Therefore, the sequence weight of B5 is higher than that of either S4 or S5. Hence, B5 wounds and removes both S4 and S5.

Table 4.15: Recording Buying Organization's Sequences

	Proposition Name	Proof Sequence	Sequence Weight
Selling Organization	p	' '	1
	S4	p→S4	0.525
	p	' '	1
	S4	p→S4	0.525
	S5	p→S4→S5	0.525
Buying Organization	B5	' '	1
	B1	B5→B1	1
	B3	B5→B3	1
	B4	B5→B4	1
Payment Authority			

In players' record sets, all entries of S4 and S5 and all affected entries whose corresponding proof sequences contain either S4 or S5, are removed. The arbiter sends a notification to selling organization, asking it to re-prove its benefit set propositions. Selling organization, however, is not able to do the re-proof. The updated information is shown in Table 4.16:

Table 4.16: Removing Entries of S4 and S5

	Proposition Name	Proof Sequence	Sequence Weight
Selling Organization	p	' '	1
	p	' '	1
Buying Organization	B5	' '	1
	B1	B5→B1	1
	B3	B5→B3	1
	B4	B5→B4	1
Payment Authority			

Finally, for payment authority, its benefit set proposition P2 is just the proposition p that has been proved by selling organization. So, there is no need for payment authority to prove P2 again. Also, because P3 equals B3, P4 equals B4, and P5 equals B5, there is no need to prove P3, P4, and P5, either.

Therefore, the accepted proof results are: B1, B3, B4, B5, P2, P3, P4, and P5. Based on these proved propositions, software arbiter executes the decision generation algorithm. Software arbiter finds no problem to construct the complete transaction state represented by letter D, which does not preserve atomicity. Therefore, software arbiter honors buying organization's request because the request reinstalls atomicity.

As illustrated by the OBI scenario above, the Wound & Remove algorithm plays a significant role in handling disputes. Although selling organization misuses a weak rule and causes conflicts, the algorithm helps software arbiter detect the problem and resolve the inconsistency among propositions. The wounding algorithm is indispensable for our arbiter architecture.

Chapter 5

A Prototype Implementation

In this section, we discuss some issues related to the implementation of a prototype system that is currently under development. The prototype implementation shows how the proposed architecture can be realized based on the client-server model.

5.1 The 3-Tier Client-server Model

The reference model of 3-tier client-server systems is composed of a set of clients, an application server and a data server [29]. Clients send service requests to the application server, which consists of application programs providing core business logic. If some requests demand accessing persistent data information, the application server then communicates with a data server, where permanent data reside.

Clients are usually Graphical User Interfaces (GUI), through which inputs and outputs can be conveniently presented. That is, system users submit their requests and view execution results through the client layer of the model. The application server is a repository of application programs, each of which deals with a particular category of client requests by executing its inherent application logic. If application programs are modeled and developed according to the object-oriented paradigm, the application server can be viewed as an object request broker. The requested services and functions are implemented by various service objects. Each object encapsulates its own application logic and has methods that can be invoked by other objects to provide various service functions. The application server, however, does not contain permanent data that could survive program execution boundaries. Hence, a data server is required to store long-term data items. The most notable form of a data server is probably a database system, which renders the access and maintenance of persistent data possible.

5.2 The Prototype Implementation

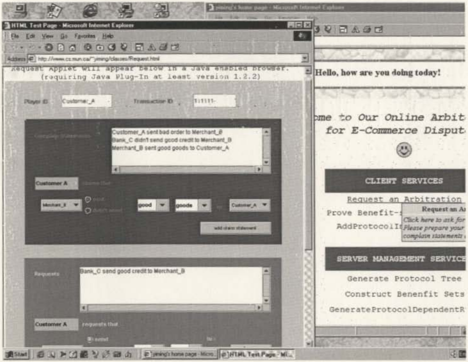
On the basis of the above reference model, our prototype system implements the architecture by mapping conceptual architecture components into their corresponding tiers in the 3-tier client-server model. Clients of the arbiter architecture are grouped into the client tier of the model. Software arbiter, rule base server and protocol tree server lie in the tier of application server. The back-end database system storing persistent information such as the protocol tree and benefit sets represents the tier of data server.

Clients of the arbiter prototype interact with the application server, e.g., software arbiter and rule base server, requesting services related to dispute handling. All services provided by the application server are encapsulated into service objects. That is, various arbitration strategies and algorithms, such as the heuristic search method and the decision generation algorithm, are realized as different kinds of application logic provided by service objects. Each object has a set of methods that can be invoked by either clients or other objects. Each method of the object corresponds to a particular service required by the normal functioning of the prototype. For instance, the findRules(char proposition) method of RuleBaseServer object provides the function of searching rules for the given proposition passed as the parameter of the method. Therefore, method invocations on service objects enable the arbiter architecture to handle disputes by providing related functions.

Whenever necessary, the service objects on the application server may in turn interact with the data server, i.e., the database system that stores the protocol tree structure and benefit sets, to retrieve requested information. As an example, ProtocolTreeServer object may access the back-end database and retrieve the benefit set of a particular player identified by playerID through its retrieveBenefitSet(int playerID) method.

5.3 Some Implementation Details

In our implementation, clients of the arbiter system are coded as Java applets. The interactions between clients and the application server are achieved through Java Remote Method Invocation (RMI) [11]. That is, the application server is developed as an RMI remote object server and client applets communicate with the server via mechanisms provided by Java RMI. If necessary, the server objects on the application server may access the data server, i.e., the database system, through JDBC [10].


HTML Test Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Address
http://www.cs.mun.ca/~yiming/classes/Request.html
Start
yiming's home page - Micro...
HTML Test Page - Mi...

```
Request Applet will appear below in a Java enabled browser.
   (requiring Java Plug-In at least version 1.2.2)
```

Player ID: Customer_A

Transaction ID: 1:1111-

Customer_A sent bad order to Merchant_B
Bank_C didn't send good credit to Merchant_B
Merchant_B sent good goods to Customer_A

Customer A

Merchant_B ○ ○ good goods Customer_A

add claim statement

Requests: Bank_C send good credit to Merchant_B

Customer A requests that

Hello, how are you doing today!

ome to Our Online Arbit
for E-Commerce Disput

CLIENT SERVICES

Request an Arbitration

Prove Benefit-

AddProtocolI

Request an A

Click here to ask for
Please prepare your
complain statements

SERVER MANAGEMENT SERVICE

Generate Protocol Tree

Construct Benenfit Sets

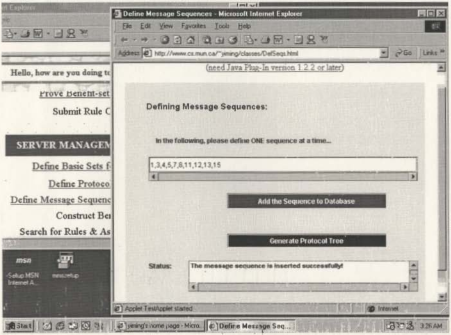
GenerateProtocolDependentR

Figure 5.1: A Snap Shot from the Implementation

The client applets are downloaded through common web browsers only when users need them. Figure 5.1 shows a Java applet interface, through which players are able to initiate a dispute. Therefore, there is no need to install software packages at the client side, thus simplifying system distribution and improving mobility of computing. In addition, maintenance of server objects becomes easy since it can be achieved on the server transparently with regard to clients. Java applets enable efficient information transmission between clients and the server because recreating or reloading entire web pages can be avoided.

Arbitration services are encapsulated as server objects whose methods can be invoked through RMI. So, the server is a repository of objects. We have applied the Factory Pattern introduced in [8] to organize these objects. That is, among all objects, there exists a main object, i.e., BrokerServer, in charge of controlling the creation of and/or access to other objects, such as SoftwareArbiter, RuleBaseServer, and ProtocolTreeServer, which provide corresponding functions of SoftwareArbiter, Rule Base Server, and Protocol Tree Server, respectively (coding details in Appendix A).

When some arbitration functions are desired, the client applets first locate the main object, i.e., the abstract factory, through the RMI naming service. Then, various other objects can be reached via this abstract factory object. If some services require the access to the back-end database system, they can do that through the interface provided by JDBC data access APIs.


Define Message Sequences - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://www.cs.mun.ca/~yiming/classes/DefSeqs.html
Go Links
(need Java Plug-In version 1.2.2 or later)
Hello, how are you doing to
Prove Benefit-set
Submit Rule C
SERVER MANAGEM
Define Basic Sets f
Define Protoco
Define Message Sequenc
Construct Ben
Search for Rules & As
Add the Sequence to Database
Generate Protocol Tree
Applet TestApplet started
Internet
Start
yiming's home page - Micro...
Define Message Seq...
3:26 AM

Defining Message Sequences:

In the following, please define ONE sequence at a time...

1,3,4,5,7,8,11,12,13,15

Status: The message sequence is inserted successfully!

Figure 5.2: An Example of Client Applets

For example, look at the client applet in Figure 5.2, which is used to define message sequences during the protocol tree building process. When the button "Add the Sequence to Database" is clicked (see codes in Appendix A), the applet first looks up the RMI registry and finds the main server object — BrokerServer, whose reference is then stored. Next, by calling the getProtocolTreeServer() method of BrokerServer, the applet can also acquire the reference to ProtocolTreeServer object. Finally, by invoking the insertMessageSequences(String sequence) method of ProtocolTreeServer object, the applet finishes inserting the message sequence into back-end database. That is, the following java statement obj.getProtocolTreeServer().insertMessageSequences(sequence) provides the service of inserting a message sequence into database.

Similarly, if the button "Generate Protocol Tree" is clicked, the related Java statement obj.getProtocolTreeServer().generateTree() should be executed, which can build the protocol tree on the server.

Because our prototype implementation is developed entirely based on Java-related techniques, it is truly platform-independent. The system is also web-enabled and thus can provide dispute handling services conveniently to EC players in diverse geographical areas. This is helpful since EC players are usually located in different areas and conduct business activities only through electronic means. The object-oriented approach adopted in the implementation also makes it easy to upgrade the system using other high-performance alternatives, e.g., the Common Object Request Broker Architecture (CORBA) [25].

Chapter 6

Conclusion and Future Work

We believe the topic of handling EC disputes is important and deserves yet more investigation. Due to the nature of EC, we think it is a promising alternative to handling EC disputes off-court and with the assistance of computer systems. Hence, in this thesis, we propose an architecture for handling EC disputes. We also describe a prototype implementation to show the architecture applicable.

We first introduce some preliminaries that are the basis of our architecture. This includes EC transactions and their important property –– atomicity, transaction protocols and their tree representations, benefit sets, and the software and human arbiters. We then propose a three-tier architecture, which consists of clients, application server and back-end database server. We show how various components can function in an orchestrated manner under such an architecture.

Because proving benefit set propositions is critical for the dispute handling process, we show how the notion of rules can be applied to assist players in proving propositions. Our focus is on rules being practically acceptable, rather than being theoretically sound, which we believe are the more realistic choices in applications. Since all these rules are not equally reliable, a measure for their reliability is essential. To this end, we introduce the concept of rule weight that reflects the reliability degree of a rule. The algorithm for rule weight calculation renders evaluating various rules possible. The application of weak rules, i.e., rules that do not have full weights, makes it possible to prove propositions that would be impossible should only full-weight rules be used. A price to be paid for such a flexibility is that some conflicts may arise. Therefore, we design the wound & remove algorithm to cope with conflicts.

In order to illustrate the arbiter architecture, we develop a prototype implementation for the architecture. The implementation is based on 3-tier client-server model and applies Java-related techniques. Functions of the arbiter architecture are realized as web services that can be easily accessed by EC players. Though the implementation is still under development, it has already shown that our architecture is feasible.

Some work deserves further study. We indicate a few in the following.

Extending the EC Protocol Model

We choose to use a simple EC protocol model introduced in section 3.3.1 because it is easier and clearer to present our basic ideas with such a model. However, practical disputes can be very complicated. For instance, some disputes, e.g., "the merchant did not send good goods to us before Jan. 15," may involve temporal aspects. Others may be about "%15 discount for club members", "free shipment within Canada", etc. These types of disputes are interesting to explore. Even within the simple model used, some complex properties, such as digital signatures and certificates usually seen in EC protocols, have not been considered. Moreover, the "good" and "bad" properties as high-level abstractions hide many details of real world situations. For instance, the bad delivery can have many forms, such as insufficient quantity, missing parts, abnormal product operation, etc. In order to handle real world disputes, these details should be considered. It is interesting to extend the currently adopted protocol model to a more complex and practical one. Consequently, new issues may arise and more work is needed.

Different Scheme of Rule Weight Calculation

In this thesis, rule search and rule weight calculation are accomplished during the initialization stage of rule base server. We adopt this strategy mainly in consideration of system performance. Applying the predefined rules and rule weights (static calculation) can provide quick response time to players. However, this is not the most accurate since proving propositions is a dynamic process where the set of proved propositions changes frequently. So far as accuracy is concerned, calculating rule weights dynamically, i.e., determining rule weights based on the "current" set of true propositions, is also an alternative worth more research. More work is needed to tackle the problem of performance degrading. Also, it may be interesting to design a hybrid mechanism that can take advantage of both static and dynamic calculations.

Practical Limitations of the Architecture

This thesis proposes an architecture that provides (semi-)automated dispute handling. Although it has been shown that the architecture is feasible with the presence of a prototype implementation, there exist some practical limitations, which may impede the immediate adoption of this system in reality.

First of all, legal issues are hard to deal with. It's clear that dispute resolutions should be backed by a legal framework that serves as an authority in making decisions such as whether the electronic evidence used is legally acceptable or not, whether the dispute handling process is followed faithfully and correctly or not, etc. This legal framework should be valid, regardless of jurisdictions in which EC participants may reside. However, at the current stage, it is not practical to find a unified, cross-border legal framework which may be entitled to apply our dispute handling system with appropriate legal effects. Therefore, currently, the proposed architecture may serve better as an estimation system helping EC participants predict possible resolutions for disputes.

Next, human factors may impede the use of our system for dispute resolution. One important principle of our dispute handling architecture is the notion of benefit set which assumes that players would not refuse to prove propositions as long as the proving does not compromise his/her interests. Yet, in reality, players may not be cooperative in proving benefit set propositions even if the proving does not harm them. They may have excuses such as 'We are too busy to do the proof'. More research efforts are needed to address those problems.

Bibliography

[1] Nabil R. Adam, Oktay Dogramaci, Aryya Gangopadhyay and Yelena Yesha. "Electronic Commerce: Technical, Business, and Legal Issues". Prentice Hall, Upper Saddle River, NJ, 1999.

[2] N. R. Adam and Y. Yesha, et al. "Electronic Commerce and Digital Libraries: towards a Digital Agora". ACM Computing Surveys, 28(4), December 1996.

[3] American Arbitration Association. Available at URL: http://www.adr.org.

[4] N. Asokan, E. Herrweghen and M. Steiner. "Towards a Framework for Handling Disputes in Payment Systems". Proceedings of 3rd USENIX Workshop on Electronic Commerce, 1998.

[5] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. Van Herreweghen, and M. Waidner. "Design, Implementation and Deployment of iKP — A Secure Account-based Electronic Payment System". Technical Report 3137, IBM Zurich Laboratory, 1999. Available at URL: http://www.zurich.ibm.com/publications.

[6] B. Cox, J. D. Tygar, and M. Sirbu. "NetBill Security and Transaction Protocol". Proceedings of the First USENIX Workshop in Electronic Commerce, pp 77-88, July 1995.

[7] Forrester Research. Available at URL: http://www.forrester.com.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, 1995.

[9] J. Gary and A. Reuter. "Transactions Processing: Techniques and Concepts". Morgan Kaufmann, San Mateo, CA, 1994.

[10] JAVA JDBC 2.0 API Documentation. Sun Microsystems, Inc., available at URL: http://java.sun.com/j2se/1.3/docs/guide/jdbc/index.html, 2000.

[11] Java Remote Method Invocation Specification, Version 1.3. Sun Microsystems, Inc., http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html, 2000.

[12] R. Kalkota and A. B. Whinston. "Frontiers of Electronic Commerce". Addison-Wesley, 1996.

[13] S. Kimbrough and R. Lee. "Formal Aspects of Electronic Commerce: Research Issues and Challenges". International Journal of Electronic Commerce, Vol. 1, No. 4, pp 11-30, 1997.

[14] Gerard Lacoste, Birgit Pfitzmann, Michael Steiner, and Michael Waidner (Eds.). "SEMPER — Secure Electronic Marketplace for Europe". Lecture Notes in Computer Science (LNCS), Volume 1854, Springer-Verlag, Berlin Heidelberg, 2000.

[15] L. Loeb. "Secure Electronic Transactions: Introduction and Technical Reference". Artech House Publishers, 1998.

[16] N. Lynch and M. Merritt and W. Weihl and A. Fekete. "Atomic Transactions". Morgan Kaufmann, San Mateo, CA, 1994.

[17] Paul May. "The Business of Ecommerce: from corporate strategy to technology". Cambridge University Press, New York, New York, 2000.

[18] Online Dispute Resolution. Available at URL: http://www.odrnews.com/whatis.htm.

[19] Indrakshi Ray and Indrajit Ray. "An Optimistic Fair Exchange E-commerce Protocol with Automated Dispute Resolution". Proceedings of EC-Web 2000, London, UK, Sep. 4-6, Lecture Notes in Computer Science 1875, Springer-Verlag, 2000.

[20] Release V2.1. Open Buying on the Internet (OBI) Technical Specifications. The Open Buying on the Internet (OBI) Consortium, http://www.openbuy.org, 1999.

[21] H. Schuldt, A. Popovici and H. J. Schek. "Execution Guarantees in Electronic Commerce". Proceedings of the 8th International Workshop on Foundations of Models and Languages for Data and Objects: Transactions and Database Dynamics (TDD'99), pp 193-202, Schloss Dagstuhl, Germany, September 1999. Lecture Notes in Computer Science (LNCS), Volume 1773, Springer-Verlag.

[22] Lei Tang. "A Set of Protocols for Micropayments in Distributed Systems". Proceedings of the First USENIX Workshop on Electronic Commerce, New York, New York, July 1995.

[23] Jian Tang and Jari Veijalainen. "A Framework for E-commerce Transaction Protocols that Support Atomicity Based Dispute Handling". Proceedings of the 3rd International Conference on Telecommunications and Electronic Commerce, Nov. 2000.

[24] Jian Tang, Ada Waichee Fu. "Secure E-commerce Transactions, Modeling and Some System Support Aspects". Proceedings of the 9th IFIP 2.6 Working Conference on Database Semantics, pp 61-75, Hong Kong, April 2001.

[25] The Common Object Request Broker: Architecture and Specification, Revision 2.4.2. Object Management Group, http://www.omg.org, 2001.

[26] J. D. Tygar. "Atomicity in Electronic Commerce". Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, pages 8-26, Philadelphia, PA, May 1996. ACM Press. .

[27] J. D. Tygar. "Atomicity versus Anonymity: Distributed Transaction Electronic Commerce". Proceedings of the 24th VLDB Conference, 1998.

[28] J. Veijalainen. "Transactions in Mobile Electronic Commerce". Proceedings of the 8th International Workshop on Foundations of Models and Languages for Data and Objects: Transactions and Database Dynamics (TDD'99), pp 208-229. Schloss Dagstuhl, Germany, September 1999. Lecture Notes in Computer Science (LNCS), Volume 1773, Springer-Verlag.

[29] Gottfried Vossen, Gerhard Weikum, and Jim Gray (Editor). "Fundamentals of Transactional Information Systems: Theory, Algorithms, and Practice of

Concurrency Control and Recovery". San Francisco, CA, Morgan Kaufmann Publishers, 2001.

Appendix A

Implementation Code Examples

The following code defines the remote interface of the main object, i.e., BrokerServer, of our arbiter server.

```
package serobj;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface BrokerServer extends Remote {
    SoftwareArbiter getSoftwareArbiter() throws RemoteException;
    // Get an instance of Software Arbiter Object
    RuleBaseServer getRuleBaseServer() throws RemoteException;
    // Get an instance of Rule Base Server Object
    ProtocolTreeServer getProtocolTreeServer() throws RemoteException;
```

```
 // Get an instance of Protocol Tree Server Object
}
```

Through the remote methods of getSoftwareArbiter(), getRuleBaseServer(), and getProtocolTreeServer(), clients are able to get instances of server objects and then invoke the desired functions provided by various methods of those objects.

Here is the remote object implementation that implements the main object, i.e., BrokerServer.

```
package serobj;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RMISecurityManager;
public class BrokerServerImp extends UnicastRemoteObject
        implements BrokerServer {
    public BrokerServerImp() throws RemoteException {
        super();
    }
    public SoftwareArbiter getSoftwareArbiter() throws RemoteException {
        SoftwareArbiterImp softwareArbiterObj = new SoftwareArbiterImp();
    return softwareArbiterObj;
}
public RuleBaseServer getRuleBaseServer() throws RemoteException {
    RuleBaseServerImp ruleBaseSeverObj = new RuleBaseServerImp();
    return ruleBaseServerObj;
}
public ProtocolTreeServer getProtocolTreeServer() throws RemoteException {
    ProtocolTreeServerImp ProtocolTreeServerObj = new ProtocolTreeServerImp();
    return ProtocolTreeServerObj;
}
public static void main(String args[]) {
    // Create and install a security manager
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
    try {
        Class.forName(“org.gjt.mm.mysql.Driver”).newInstance();
        // Load the database driver since some server objects
        // may need to access the backend database
        System.out.println(“Database driver loaded...”);
        try {
                BrokerServerImp serobj = new BrokerServerImp();
                // Bind this object instance to the “Service-Broker-Server”
                // The RMI registry name of our main
                // server object is:  Service-Broker-Server
                Naming.rebind(“Service-Broker-Server”, serobj);
                // After this RMI registry name binding, Service-Broker-Server
                // can be located by clients later
                System.out.println(“E-Commerce Arbitration Server in Service!”);
            } catch (Exception e) {
                System.out.println(“BrokerServerImp err: ” + e.getMessage());
                e.printStackTrace();
            }
        } catch (Exception E) {
            System.err.println(“Unable to load database driver...”);
            e.printStackTrace();
        }
    }
}
```

In the following, we present some representative APIs defined in the remote interfaces of core server objects, namely, SoftwareArbiter, RuleBaseServer, and ProtocolTreeServer.

```
public interface SoftwareArbiter extends Remote {
    RuleBaseServer getRuleBaseServer() throws RemoteException;
    // Get an instance of Rule Base Server Object
    ProtocolTreeServer getProtocolTreeServer() throws RemoteException;
    // Get an instance of Protocol Tree Server Object
    String submitComplaint(String complaint, int transID) throws RemoteException;
    // Submit the complaint statement and save it in database
    String submitRequest(String request, int transID) throws RemoteException;
    // Submit the request statement and save it in database
    String submitDirect(String[] proofResults, int playerID) throws RemoteException;
    // Submit the proof results for benefit set propositions of a player
    // The proof results are done directly without applications of rules
    String submitRule(String[] proofSequences, int playerID) throws RemoteException;
    // Submit the proof results for benefit set propositions of a player
    // The proof results are done via applications of rules
    String woundRemove(String[] proofSequences) throws RemoteException;
    // The wound and remove algorithm used to handle conflicts
    String insertRecordSetEntry(String[] entry, int playerID) throws RemoteException;
    // Insert a record set entry into the log
    String[] checkConflicts() throws RemoteException;
    // Check whether or not there are some conflicts in the record sets
    String removeRecordSetEntry(int entryID, int playerID) throws RemoteException;
    // Remove the record set entry identified by entryID
    String generateDecision(String[] proofResults) throws RemoteException;
    // The decision generation algorithm
    String notifyHumanArbiter(String notification) throws RemoteException;
    // Notify human arbiter asking for assistance
    String login(String user, String pwd) throws RemoteException;
    // Check login information for a user.
}
public interface RuleBaseServer extends Remote {
    SoftwareArbiter getSoftwareArbiter() throws RemoteException;
    // Get an instance of Software Arbiter Object
    ProtocolTreeServer getProtocolTreeServer() throws RemoteException;
    // Get an instance of Protocol Tree Server Object
    String[] findRules(char proposition) throws RemoteException;
    // Find applicable rules for a proposition
    String heuristicSearch() throws RemoteException;
    // Heuristic search algorithm which generates rules
    double calculateEdgePro(String[] edge) throws RemoteException;
    // Calculate the edge probability of an edge in the protocol tree
    double calculatePathPro(String[] path) throws RemoteException;
    // Calculate the path probability of a path in the protocol tree
    double calculateRuleWeight(String[] rule) throws RemoteException;
    // Rule weight calculation algorithm
    String login(String user, String pwd) throws RemoteException;
    // Check login information for a user
}
public interface ProtocolTreeServer extends Remote {
    SoftwareArbiter getSoftwareArbiter() throws RemoteException;
    // Get an instance of Software Arbiter Object
    RuleBaseServer getRuleBaseServer() throws RemoteException;
    // Get an instance of Rule Base Server Object
    String executeSql(String sql) throws RemoteException;
     // Submit a SQL statement to the database for execution
     // The SQL statement cannot be a select type
    String insertPlayerSet(String player) throws RemoteException;
    // Insert a player into the player set table
    String insertContentSet(String contentItem) throws RemoteException;
```

```
// Insert an item into the content set table
String insertPropertySet(String attributeItem) throws RemoteException;
 // Insert an attribute into the property set table
String insertMessages(String message) throws RemoteException;
// Insert a message into the message table
String insertMessageSequences(String sequence) throws RemoteException;
// Insert a sequence into the message sequence table
String[] selectPlayerSet() throws RemoteException;
// Retrieve the player set
String[] selectContentSet() throws RemoteException;
// Retrieve the content set
String[] selectPropertySet() throws RemoteException;
 // Retrieve the property set
String generateTree() throws RemoteException;
// Genearte the protocol tree structure and save it in database
String constructBenefitSet() throws RemoteException;
// Construct players' benefit sets based on the protocol tree
String[] retrieveBenefitSet(int playerID) throws RemoteException;
// Retrieve a player's benefit set
String login(String user, String pwd) throws RemoteException;
// Check login information for a user
```

```
}
```

The following codes are executed to store the sequence input by users into the corresponding database table.

```
String sequence=jTextField_Sequence.getText();
try {
    jTextArea_Status.append(“Contacting the server... ”);
    BrokerServer obj = (BrokerServer)Naming.lookup(“//” + getCodeBase().getHost() +
                                                        “/Service-Broker-Server”);
    // Look up the RMI registry and find the main server object: Service-Broker-Server
    message = obj.getProtocolTreeServer().insertMessageSequences(sequence);
    // Invoke the insertMessageSequences(String sequence) method of
    // ProtocolTreeServer object which can be located through BrokerServer
    jTextArea_Status.append(message);
    // Display in the status bar the message of execution results returned from the server
} catch (Exception ex) {
    jTextArea_Status.append(“\nService-Broker-Server access exception: ”
                              + ex.getMessage());
    ex.printStackTrace();
    // Handle exceptions
}
```

As listed previously, the method insertMessageSequences(String sequence) of ProtocolTreeServer object inserts the message sequence that is passed as the parameter into the corresponding database table, therefore, the Java statement in the above code obj.getProtocolTreeServer().insertMessageSequences(sequence) can insert the message sequence submitted by users into database.

Since the insertMessageSequences(String sequence) method needs to access the database, codes in the method apply JDBC APIs.

```
public String insertMessageSequences(String sequence) {
    String message;
    message=“”;

    try {
        Connection  Conn = DriverManager.getConnection(
            “jdbc:mysql://heron/ecdata?user=ecmun&password=ec111”);
        // Create the database connection according to JDBC driver requirements
        // In our implementation, the database is MySql, version 3.21
        System.out.println(“Connection established!”);
        try {
            Statement Stmt = Conn.createStatement();
            String sqlStatement="insert into Sequences values ('" + sequence + "')";
            // Format the SQL statement, which does the insertion of sequence
            Stmt.executeQuery(sqlStatement);
            // Execute the SQL statement in the database
            Stmt.close();
            Conn.close();
            return message+ "Sequence insertion OK... ";
        } catch (SQLException E) {
            // Handle exceptions
            System.out.println("SQLException: " + E.getMessage());
            System.out.println("SQLState:     " + E.getSQLState());
            System.out.println("VendorError:  " + E.getErrorCode());
        }
    } catch (SQLException E) {
        // Handle exceptions
        System.out.println("SQLException: " + E.getMessage());
        System.out.println("SQLState:     " + E.getSQLState());
        System.out.println("VendorError:  " + E.getErrorCode());
    }
    return message+ "Sequence insertion failed! ";
}
```