

ABSTRACT

CHENG, ZHENGANG. Verifying Commitment Based Business Protocols and their Compositions: Model Checking using Promela and Spin. (Under the direction of Professor Munindar P. Singh and Professor Mladen A. Vouk).

A protocol-oriented approach of modeling and enacting business processes and workflows has been developed recently that offers advantages in terms of supporting the autonomy and heterogeneity of business partners and the reconfigurability of their process. Importantly, protocols are described using commitments, map to the individual computation of the participating roles, and can be composed to yield more complex protocols. However, verifying that the protocols, especially composed protocols, fully satisfy appropriate correctness properties remains an open problem.

This dissertation presents a novel way to model business protocols in terms of commitments involved and the constraints for protocol composition. The correct composition of a business process can be expressed via individual protocol definitions and their composition constraints, thereby enabling the verification of large processes. Importantly, as part of the verification process, protocols are translated into the language Promela, which makes them amenable to analysis and verification using the model checker Spin. As a result many important properties of business protocols and their compositions into partial and full workflows can be verified, and improved protocols can be produced.

The contribution of this dissertation is in providing a generalized mechanism for modeling commitments, formulating and verifying properties related to commitments. In fact, the results are applicable to a wide range of processes and related protocols, such as scientific discovery processes and workflows.

**Verifying Commitment Based Business Protocols and their Compositions:
Model Checking using Promela and Spin**

by

Zhengang Cheng

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh

2006

Approved By:

Dr. Laurie Williams

Dr. Peter R. Wurman

Dr. Munindar P. Singh
Cochair of Advisory Committee

Dr. Mladen A. Vouk
Cochair of Advisory Committee

To my wife and parents
for their support, understanding, and love

Biography

Zhengang Cheng was born in a small village in Yong Kang County, Zhejiang province, P.R. China on May 4, 1973. On September 1991, he left home for the first time to a big city to attend the University in Nanjing. He received his M.S. and B.S. in Computer Science from Nanjing University of Aeronautics and Astronautics in 1995 and 1998, respectively. Zhengang joined the Huawei company in Shenzhen for 4 months after graduation.

In August 1998, Zhengang arrived in the United States and started his graduate studies at North Carolina State University. Zhengang's research involves software agents, business protocols and composition, scientific workflow, focusing on protocol modeling and verification. During his graduate study, he also worked as a summer intern at Viewcast.com, Nortel Network, and IBM at Research Triangle Park.

Acknowledgements

It is a difficult task to guide a student like me who has diverse interests and has difficulty focusing on the most important things for a period of time. Special thanks to Professor Munindar P. Singh and Professor Mladen A. Vouk for their continued financial and emotional support through so many years, and with great encouragement and guidance to finish this long and difficult journey. I am always indebted to their great giving and guidance.

I would like to extend my appreciation to Dr. Laurie Williams and Dr. Peter Wurman for their valuable discussions and comments regarding my research.

Also I thank my colleagues at the Service-Oriented Computing lab Amit Chopra, Nirmal Desai, Yathi Udupi, Bin Yu, Jie Xing, and Feng Wan for their great comments and help. It was a great pleasure to work with Sangeeta Bhagwanani, Sandeep Chandra, Pierre Moullem, and Brent Vincent Marinello in the SDM Project. I also thank Marhn H Fullmer for solving my equipment problems.

Finally I want to express my utmost appreciation to my wife and my parents for their love, support, patience, encouragement, and understanding throughout my long graduate studies.

This work was supported in part by the DOE SciDAC grant DE-FC02-01ER25484, NSF grants IIS-9529179 and IIS-9624425, IBM Shared University Program, and StrikeIron Inc.

Contents

List of Figures	vi
------------------------	-----------

List of Tables	vii
-----------------------	------------

1 Introduction	1
1.1 Service-Oriented Computing	1
1.2 Application Domains for Service-Oriented Computing	3
1.2.1 Business Processes and Business Protocols	3
1.2.2 Scientific Workflows	4
1.3 Motivations	6
1.4 Organization	7
2 Technical Framework	8
2.1 OWL-P	8
2.1.1 OWL-P Composition Profile	10
2.1.2 OWL-P Rules	11
2.1.3 Commitments	11
2.2 Example Protocols	13
2.2.1 Purchase Protocols	13
2.2.2 Astrophysics Workflow Example	17
2.3 Model Checking	19
2.3.1 Linear Time Logic	20
2.3.2 Spin and Promela Primer	21
2.3.3 Verifying Composed Services	22
3 Translating Business Protocols to Promela	24
3.1 Role Skeletons in Promela	25
3.2 Composite Protocols in Promela	30
3.3 Modeling Commitments in Business Protocols	35
3.3.1 Operations on Commitments	35
3.3.2 Translating Commitments into Promela	36
3.3.3 Commitment Modeling in Role Process	40

3.3.4	Commitment Operation Example Scenarios	41
3.4	Correctness of the Translation	44
4	Verification for Business Protocol Properties	45
4.1	Specifying Property Formulae in Spin	45
4.2	General Properties	47
4.2.1	Deadlock and Livelock Freedom	47
4.2.2	Termination of Role Processes	47
4.2.3	Commitments in Good States	47
4.2.4	Unconsumed Messages in Channel	48
4.3	Protocol-Specific Properties	48
4.3.1	Ensure Goods if Pay	49
4.3.2	Ensure Pay if Goods Shipped	49
4.3.3	No Shipment if Reject	49
4.3.4	No Payment if Reject	50
4.4	Verifying Commitment Properties	50
4.5	Progressive Refinement	50
4.5.1	Missing Axioms	51
4.5.2	Conflicting Axioms	51
5	Discussion and Conclusion	52
5.1	Applicability	52
5.1.1	Applicability in Business World	52
5.1.2	Applicability in Scientific Workflow	53
5.2	Limitations and Possible Extensions	54
5.2.1	Expressiveness of Composition Axioms	54
5.2.2	Modeling Commitment Propagation	55
5.2.3	Modeling Repetitive Process	55
5.2.4	Supporting Multiple Instances of a Role	56
5.3	Literature	56
5.4	Conclusion and Future Directions	58
	Bibliography	59

List of Figures

2.1	Basic OWL-P ontology	9
2.2	Commitment lifecycle	13
2.3	Order protocol	14
2.4	Payment protocol	14
2.5	Shipping protocol	15
3.1	The steps of translating OWL-P to Promela and verifying properties	24
3.2	The common commitment process	37
3.3	Participating roles in the purchase protocol	42

List of Tables

2.1	Composition Axioms in Purchase Protocol	16
2.2	Semantics of LTL	21
3.1	Mapping of role skeleton rule for ρ_1 sending and ρ_2 receiving message . . .	28
3.2	Mapping of role skeleton rule for ρ_1 End	29
4.1	Some generic formulae of LTL	46
4.2	Updated Composition Axioms in Purchase Protocol	51

Chapter 1

Introduction

As web service standards and tools are maturing, service composition is gaining attention for companies to automate and reduce cost. Although more companies are offering information and services online, most current services available online are mainly oriented towards human browsing, navigation, and invocation. Web services are oriented for program access, and geared for easy integration of services across different enterprises.

1.1 Service-Oriented Computing

In order for services from different enterprises to be able to interact with each other, an interoperable service environment is required. We envision in the near future such a service environment will mature. Current web services provide a foundation for such an interoperable service environment.

In a web service environment, service providers can publish their service in a central registry. Service consumers can use the retrieved services to construct new services. Service composition provides an opportunity for a company to offer their new value-added services by integrating their business process with business partners.

In the scientific workflow field, a scientist may be more interested in integrated tools and data from other organizations. Service composition provides the flexibility required in scientific investigation to model and enact scientific workflows. Scientific workflow usually involve more data and are more computationally intensive than business workflows. Scientific workflow tools usually pay more attention to easy automation, in order to study

large sets of data. For example, a biologist might want to investigate dozens or hundreds of protein sequences against huge databases like the human genome database possibly using the Blast program.

Such a service environment creates the opportunity for service providers to deliver value-added and integrated services to more potential service consumers. At the same time, new services can be constructed at different levels of granularity from services offered by other enterprises. From the point view of service consumers, they can find needed services quickly and integrate to their internal process or workflow easily, thus improving efficiency and reducing costs.

The need and opportunity of service composition is obvious in both the scientific community and the business world. Although many research issues in service composition are investigated, a general service composition platform and environment is not yet a reality. Issues such as composability, compatibility, conformance and substitutability, correctness, synchronization, and coordination in service composition are not yet thoroughly investigated. For example, conformance and consistency rules should be checked to find errors as early as possible in the workflow construction process.

Obviously the service composition environment will be distributed and heterogeneous in terms of service platform such as operating systems hosting the service and the programming languages used to develop the service. Each enterprise or organization will retain full control over its data and services, and at its own discretion grant rights to use its service to some designated groups of users. The organization can utilize the most appropriate technologies to its unique business.

Each service offered in the environment is autonomous and self-contained. For the service consumer, the service can be considered a black box, with only the service definition available from the service registry to interpret the functionalities and properties of the service.

To maximize the flexibility and interoperability of service composition, and minimize the integration cost, we assume that all services conform to some minimal interoperable standards. The web service platform is a good foundation for such an interoperable environment.

In such an environment, the traditional workflow systems cannot work well, since there

is no central authority to coordinate synchronization of databases or services. It is the burden of the service consumer or service composer to oversee the smooth execution of composed service. There is no central service engine as an equivalent of workflow engine in a traditional workflow system.

We identify the following major players in the service composition environment.

- Service Provider: Providing services and publishing those services in a service registry.
- Service Consumer (Requester): Utilizing services provided by others: they can find desired services through a service registry.
- Service Composer: Utilizing other's services and own services to create new services and publishing them. In this sense, a composer is a provider and a consumer at the same time.
- Service Registry: Provides (public) facilities for service provider and service consumers to find each other.

1.2 Application Domains for Service-Oriented Computing

1.2.1 Business Processes and Business Protocols

Companies conduct business through business protocols. In order to utilize the opportunity provided in service-oriented computing, a company needs to design its interaction protocols compatible with services or workflow from its business partners.

Designing business processes for open settings is notoriously complex. Business partners are autonomous and heterogeneous, the configuration of a process can change dynamically, and exceptions frequently arise because of real-world or organizational problems. To address these challenges, we are pursuing an interaction-oriented multiagent approach for modeling and enacting business processes. Our approach centers around specifying the interactions among autonomous, heterogeneous partners. A (*business*) *protocol* is a public specification of an interaction. A protocol typically deals with a well-defined business

purpose such as payment or order placement. Moreover, in our treatment, a protocol is specified in terms of the commitments among the different parties. Doing so reduces the emphasis on the rigid sequence of actions that the different parties must take, and enables flexible processing based on the commitments.

A protocol serves as an interface, specifying only the key aspects of interactive behavior, not how the partners' software components are implemented. Each participant's perspective of the interaction, known as a *role*, can be extracted from a protocol. An *agent* is a computational entity that represents a business partner. An agent plays one or more roles and incorporates the private *business policies* of the partner it represents. A business process is enacted when the appropriate agents interact. Modeling interactions based on commitments facilitates loose coupling among the parties. Loose coupling promotes autonomy and heterogeneity, and yields a system that handles exceptions and opportunities perspicuously. By contrast, traditional approaches specify or derive control and data flows over services. Such representations operate at a low level, specifying the actions that different parties must take, and result in a tight coupling among business components.

1.2.2 Scientific Workflows

With the availability of more powerful machines, and large data such as the human genome collected or simulated, scientists find the need to utilize the work of others in terms of tools generated and data available. Those tools and data can be public and open for use by everyone like the data published NIH websites. Other tool and data can be commercial or protected for various considerations. It is unrealistic to expect the scientists to reinvent the wheels. That is, it is imperative to use the data and tools available, so scientists can be more productive and devote more energy into the main investigation and research.

Scientific workflow is a process where the scientists have linked and integrated analysis tools and data from different organizations or companies. Scientific Data Management (SDM) [3] research is devoted to streamlining the process of scientific research, so that repetitive works can be taken care of automatically, and the process can be easily modified for different parameters or situations. The Kepler project [1] is an endeavor of scientists from various research area like in Biology, Fusion, Combustion, Astrophysics to achieve

such a goal. Kepler is based on the work of Ptolemy project [2] from UC Berkeley.

Scientific research is often computational and data intensive. It needs to have access to the most powerful machines in the world, and may generate huge amount of data in the scale of tera-bytes. And because those computational resources and data storage facilities are expensive, and such research might be very sensitive, there is always the great need of security. The system has to ensure only the correct person access the correct sets of data.

Those resources only have limited availability to the scientists. First they are so expensive that only very few machines are available, most only available in national research labs like Oak Ridge National Laboratory, Princeton Plasma Physics Laboratory, and Lawrence Livermore National Laboratory. As scientists are scattered around universities and research organizations around the states or even the world, they have to access the system remotely. Second, since so many people want to use those supercomputers, their availability is very limited. Scientists want to make full use of their allocated time-slots and processors. Strong security mechanisms guard the facility. Originally these resources were accessible via SSH, which encrypts the communication with the supercomputers to protect sensitive information. Recently a random password mechanism has been implemented, where users input a random passcode generated by a fob along with some password. In this way the password is changed about every 30 seconds, greatly enhancing the security.

The following is a list of characteristics of the scientific workflow environment.

1. Heterogeneous: The data and tools likely belong to different organizations. The machine platforms are different. For machines providing similar functionalities, may have a slight difference in parameters. For example, the "date" function on the Cray supercomputer is different from the "date" function on a Linux cluster.
2. Security: All those supercomputers or clusters process sensitive information. The access to the systems is highly secured, which limits the ways to utilize the system.
3. Lack of modularity: There are many different executables in a host. However when we want to use them in a workflow, they must be encapsulated.
4. Huge data: The amount of data is usually huge. Due to limited storage space, there is a need to transfer the resulting data from the supercomputer to local storage facility

for storage or further processing needs. Moving this huge amount of data takes a long time due to the network speed limit.

With scientific workflows need to integrate several diverse services, they need a facility for modeling correct workflows, and ensuring their correctness. The results of this dissertation can help there.

1.3 Motivations

As a design abstraction, protocols are akin to software components. Protocols are inherently modular; they can be created ahead of time and stored in libraries. Particular protocols may then be selected and composed to support a desired business application. Desai et al. recently developed a framework, OWL-P, which supports the specification and composition of business protocols [11]. Composition is achieved by specifying a composition profile, which is a set of operational dependencies between the constituent protocols. This dissertation extends the OWL-P framework by enabling automated verification of protocol compositions. Verifying protocol composition is nontrivial as the constituent protocols may be complex, and their composition may exhibit subtle, potentially undesirable properties. Moreover, it is unlikely that a designer would be able to specify the correct dependencies at one shot. Like other intellectual efforts, protocol composition is best performed iteratively, through progressive refinement.

As service is more prevalent, businesses want leverage on the benefits of a service-oriented environment. More business processes will interact with other businesses within or outside of the company. This forms a network of interacting business processes. Each business process is following some business protocol. The composition of small business protocols enables the larger business process.

Similarly, in scientific research, there is the need to integrate disparate data and tools in research investigation, thus combining simply services into a sophisticated research process. The composition of protocols allows more flexibility in performing a business process, but it also introduces more complexity. We need to make sure that there are no loopholes. Most businesses involve exchange of money, goods, and services. Each partic-

ipating party has special obligations and benefits in the process. It is important that those commitment to be well observed.

This dissertation makes the following contributions:

1. Model commitments between roles in a business protocol.
2. Mechanically translate protocol specifications from OWL-P into Promela, which is the program specification language for the model-checker Spin [20].
3. Map dependencies in a composition profile to a corresponding Promela model, which can be checked for various temporal properties.
4. Identify subtle side-effects produced by the dependencies.

This dissertation describes a way of verifying the correctness of individual protocols and composed protocols.

1.4 Organization

The rest of the dissertation is organized as follows. Chapter 2 describes the relevant details of the OWL-P framework, and a primer on Promela. Chapter 3 presents the translation of OWL-P specifications and composition profiles into Promela. Chapter 4 looks at the model-checking of compositions, and the commitments involved. Chapter 5 discusses the relevant literature and describes some important research themes and some future research directions.

Chapter 2

Technical Framework

This chapter describes our framework for specifying and enacting business processes based on protocols. The framework includes the protocol specification language OWL-P, which incorporates an ontology for protocols expressed in OWL [5]. We give a short discussion in commitments in Section 2.1.3, which is used to model the proper behavior of participants in a business protocol. We use a purchase protocol described in Section 2.2 as our example business protocol. Promela as the language for model checking is introduced in Section 2.3.2.

2.1 OWL-P

OWL-P (OWL for Protocols and Processes) [5, 11] is a practical framework and an associated language for specifying, combining, and enacting commitment protocols. OWL-P uses OWL (Web Ontology Language) to specify protocol and protocol composition constructs. An OWL-P specification identifies roles that participate in the protocol, the messages that are exchanged (with the meanings of the messages in terms of commitments that are created), and a set of rules that constrain the set of runs of the protocol by specifying ordering, data flow, and other constraints.

At run time, when a protocol is enacted, the roles in the protocol are adopted (i.e. played) by different agents. The agents are expected to respect the protocol but have a lot of flexibility in how they choose their actions. For this reason, we think of agents as long-lived computational entities that perceive, reason about, and act in their environment.

Importantly the agents communicate with each other.

Figure 2.1 shows the OWL-P ontology with concepts as nodes and their properties as edges (the end of an edge marked with a rectangle identifies the domain of a property). A protocol is specified using implication rules, as in the Semantic Web Rule Language (SWRL) [22]. To elucidate protocol specification, we use the protocols involved in a typical purchase process detailed in Section 2.2.

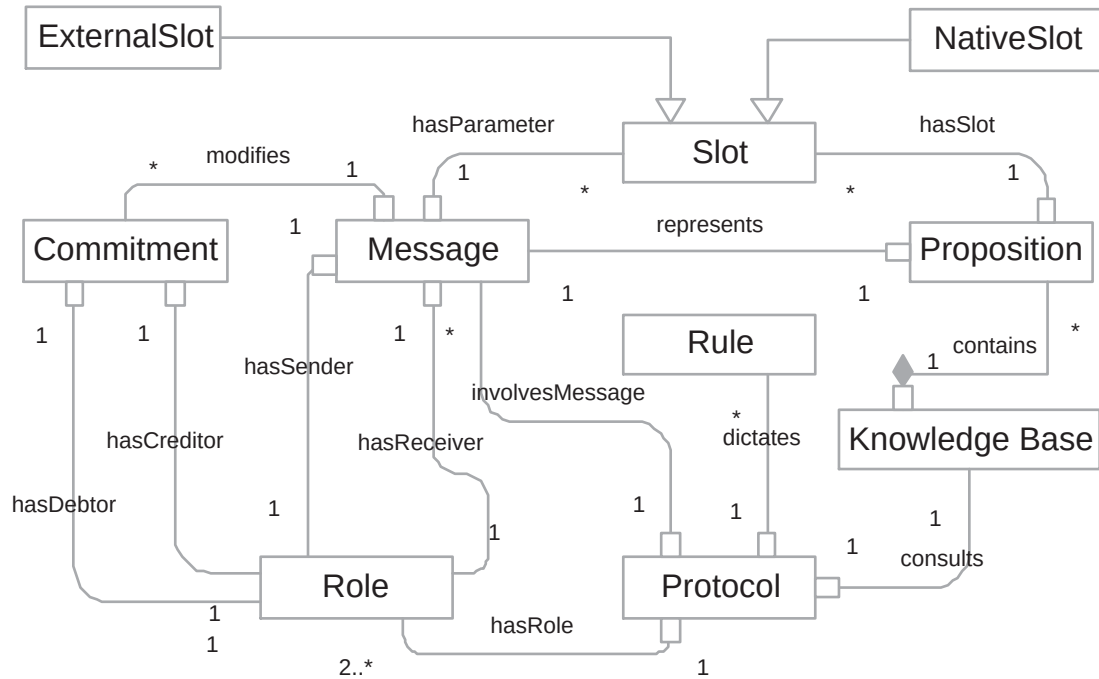


Figure 2.1: Basic OWL-P ontology

A notable feature of OWL-P is that it annotates messages exchanged by various participants in a protocol in terms of how commitments are created and discharged among participants. Representing commitments enables a pragmatic notion of compliance verification. In addition, agents can reason about their future actions depending on their existing or prospective commitments.

Roughly, a commitment corresponds to an obligation from a debtor to a creditor about a particular condition. The formal definition of commitment is in Section 2.1.3. Commitments support operations such as create, discharge, and others [27].

A *Message* has a sender, a receiver, and one or more parameters, termed *Slots*. The

semantics of a message is given by its effects on commitments, expressed via commitment operations. The slots function as variables. A slot is *defined* when it is assigned a value and is *used* when its value is assigned to another slot (or is referenced within a predicate. A *local slot* of a protocol is defined within the protocol. An *external slot* is assigned a value produced by another protocol.

A *protocol* dictates rules that govern the interaction and consults a *knowledge base*, which consists of a set of *propositions*. Propositions are represented as slotted facts in a rule-based system. Propositions express active commitments, domain facts, and messages sent or received.

2.1.1 OWL-P Composition Profile

A composition profile describes the relationships among protocols that must be preserved when composing protocols. These relationships are called *axioms* in OWL-P terminology. There are four kinds of axioms as described below.

- **Role Definition.** A role definition axiom defines composed protocol in terms of roles in the protocols being composed. The properties of a role definition axiom are *define* and *unify*. The former specifying the name of the role in the composed protocol and the latter specifying a role in one of the composed protocols. Each role definition axiom has exactly one value for the *define* property.
- **Data Flow.** A data flow axiom provides the bindings for external slots in protocols, since an external slot's value has to be bound outside the scope of the protocol in which it is declared to be external. The properties of a data flow axiom are *definition* and *usage*, the former stating where the value of a slot will be bound and the latter stating where it will be used (as an external variable). Each data flow axiom has exactly one value for the *definition* property.
- **Implication.** An implication axiom is used to denote propositions in different protocols that have the same meaning. The properties of an implication axiom are *antecedent* and *consequent*, the value of the former property logically implies the value of the latter property.

- **Event Order.** An event order axiom specifies temporal ordering among messages in the protocols being composed.

The composition profile is developed by a protocol designer. A composed protocol can be composed further like any other protocol.

2.1.2 OWL-P Rules

OWL-P specifies protocols declaratively, using rules of the form *antecedent* $\rightarrow$ *consequent*. To be compatible with OWL syntax, the Semantic Web Rule Language (SWRL) has been adopted in OWL-P to specify rules. Antecedents check the knowledge base for the existence or absence of propositions. Consequents effect message transfers and add related propositions to the knowledge base. In effect, rules fire based on the history of the interaction as captured by propositions in the knowledge base. One of the most important features of OWL-P, in contrast with traditional process models such as workflows, is the distributed nature of OWL-P, where each participant of a protocol is a peer, unlike in a client-server system. The knowledge base, therefore, cannot be centralized. Further, a distributed enactment framework implies that no participant is guaranteed to be able to observe all messages (and significant events) in the protocol. The OWL-P algorithm for generating role skeletons incorporates this feature of distributed systems and is sound and complete with respect to the set of runs it generates.

2.1.3 Commitments

A commitment is an obligation from a debtor x to a creditor y about a particular condition p . A commitment has the following two basic forms.

- *Unconditional commitment* $C(x, y, p)$. A commitment whose condition p will be brought about unconditionally. Here x and y are known as the debtor and creditor of the commitment, respectively. Such a commitment is called a *base-level* commitment. For example, $C(\text{buyer}, \text{seller}, \text{pay})$ denotes that the buyer promises to pay the seller.

- *Conditional Commitment* $CC(x, y, p, q)$. A commitment whose condition q will be brought about if the precondition p becomes true. This means that the base-level commitment $C(x, y, q)$ comes into being when the precondition p of the conditional commitment holds. For example, $CC(\text{buyer}, \text{seller}, \text{ship}, \text{pay})$ denotes that the buyer promises to pay the seller if the goods are shipped to him.

The precondition p can be expressed as a compound predicate which could embed other commitments as needed. This allows us to define commitments recursively and to specify complex obligation dependencies. For example, we could define a commitment $CC(\text{buyer}, \text{seller}, C(\text{seller}, \text{buyer}, \text{ship}), \text{pay})$ meaning that the buyer promises to pay the seller if the latter promise to ship the goods. This commitment differs from the previous conditional commitment in that the payment depends only on the promise of shipping and may be performed before the actual shipping.

We require that for any inner commitment appearing in precondition p , the creditor must be the debtor of the immediate outside commitment. To illustrate this, if a commitment has form $CC(x, y, \dots C(w, v, q) \dots, p)$, then $x = v$. The motivation is that the debtor x 's commitment is conditioned on another party doing something for x .

Commitments support several operations that combine to capture mutual and multiparty scenarios. For the sake of simplicity, this dissertation is limited to four main operations. The four operations, *create*, *update*, *discharge*, and *cancel*, drive the lifecycle of commitments. A commitment is initially created when an agent makes a promise to another agent. If the commitment has been fulfilled, e.g., the conditions have become true, then the commitment is discharged. However, before the commitment is discharged, the agents involved can possibly update the commitment. The update operation gives flexibility in manipulating agents' context to react to any potential requirement changes or exceptions. Agents can also cancel their commitments, e.g., to accommodate exceptions. However, to cancel a commitment, agents usually face penalties that compensate for whatever inconsistencies that they may have introduced by canceling their commitment (upon which, presumably the creditor dependent). Figure 2.2 shows the state diagram of the commitment lifecycle.

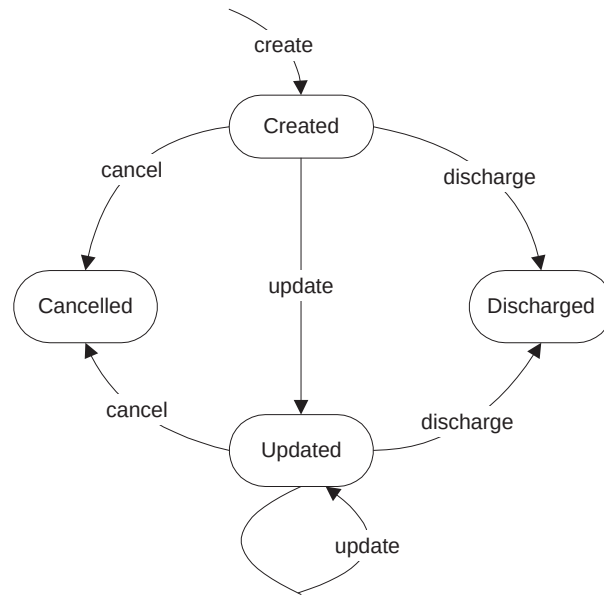


Figure 2.2: Commitment lifecycle

2.2 Example Protocols

We will use purchase process as our example for modeling and verification. An typical scientific workflow is described as well.

2.2.1 Purcase Protocols

A protocol defines how participants should interact with each other. It is similar to human languages that allow the participants of a business process to follow some rules of conversation to effectively work with each other. Figures 2.3, 2.4, and 2.5 show three example protocols: *Order*, *Payment* and *Shipping*, respectively. These protocols are a great running example for the present research because of the importance of the purchasing scenario. However, all our results are generic and apply to all protocols based on commitments. Different roles participate in the protocols. For example, in *Order* protocol, the roles are *Buyer* and *Seller*.

Leading and trailing asterisks (*), as in **amount**, identify an external slot. Consider the *Order* protocol. The buyer requests a quote for an item; the seller responds with a

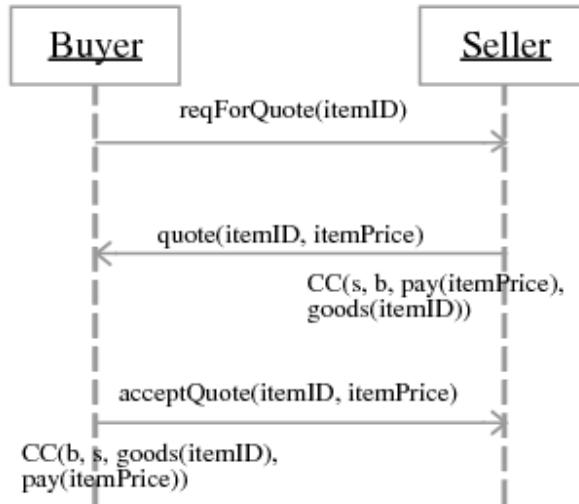


Figure 2.3: Order protocol

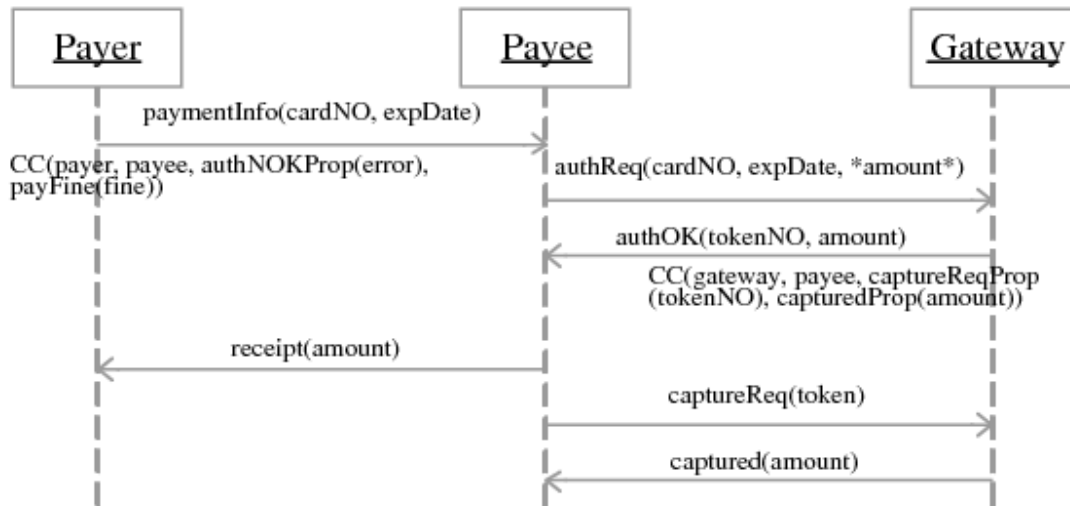


Figure 2.4: Payment protocol

quote. The semantics of quote is that it creates a commitment from the seller to the buyer guaranteeing delivery if the buyer pays the quoted price. In this scenario, the buyer accepts the quote. The semantics of accept is to create a commitment from the buyer to the seller to pay the quoted price if it receives the requested item. These rules are interpreted operationally as controlling the communicating parties and thus impose a temporal order. Thus, the propositions in the ontology correspond to fluents, e.g., as in action logics [19]. Below are the rules for *Order* in the “antecedent $\Rightarrow$ consequent” notation.

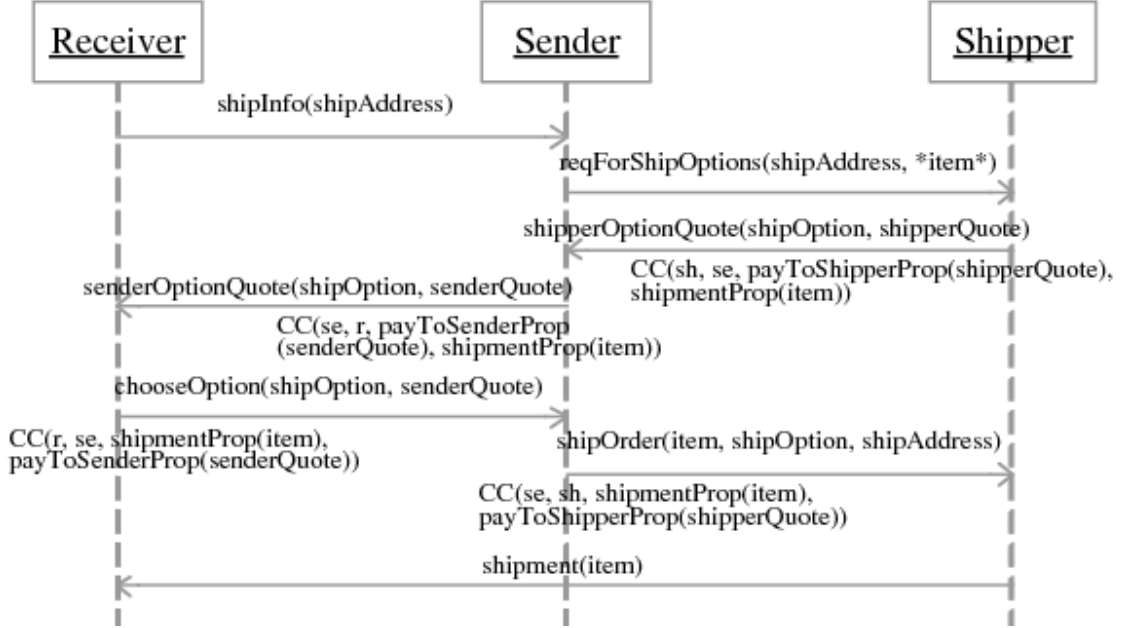


Figure 2.5: Shipping protocol

Order1. $\text{start} \Rightarrow \text{reqForQuote}(?itemID)$

Order2. $\text{reqForQuoteP}(?itemID) \Rightarrow$

$\text{quote}(?itemID, ?itemPrice) \wedge \text{CC}(S, B, \text{pay}(?itemPrice), \text{goods}(?itemID))$

Order3. $\text{quoteP}(?itemID, ?itemPrice) \wedge \neg \text{rejectQuote}(?itemID, ?itemPrice) \Rightarrow$

$\text{acceptQuote}(?itemID, ?itemPrice) \wedge \text{CC}(B, S, \text{goods}(?itemID), \text{pay}(?itemPrice))$

Order4. $\text{quoteP}(?itemID, ?itemPrice) \wedge \neg \text{acceptQuote}(?itemID, ?itemPrice) \Rightarrow$

$\text{rejectQuote}(?itemID, ?itemPrice) \wedge \text{CC}(B, S, \text{goods}(?itemID), \text{pay}(?itemPrice))$

Order5. $\text{acceptQuoteP}(?itemID, ?itemPrice) \Rightarrow \text{end}$

Order6. $\text{rejectQuoteP}(?itemID, ?itemPrice) \Rightarrow \text{end}$

The knowledge base contains *start* at the start of a protocol. Here, *reqForQuote*, *quote*, *acceptQuote*, and *rejectQuote* are OWL-P messages, and their corresponding propositions are *reqForQuoteP*, *quoteP*, *acceptQuoteP*, and *rejectQuoteP*. The commitment conditions *pay* and *goods* are also modeled as propositions. Parameters *?itemID* and *?itemPrice* are native slots. The *?itemID* in the first rule is not assigned any value by the antecedent, im-

Table 2.1: Composition Axioms in Purchase Protocol

- | |
|---|
| 1. <code>roleDefinition (define: Purchase.customer , unify: Order.buyer ,
unify: Shipping.receiver , unify: Payment.payer)</code> |
| 2. <code>roleDefinition (define: Purchase.merchant , unify: Order.seller ,
unify: Shipping.sender , unify: Payment.payee)</code> |
| 3. <code>dataFlow (definition: Order.itemID , usage: Shipping.item)</code> |
| 4. <code>dataFlow (definition: Order.itemPrice , usage: Payment.amount)</code> |
| 5. <code>implication (antecedent: Shipping.shipmentProp ,
consequent: Order.goods)</code> |
| 6. <code>implication (antecedent: Payment.authOKProp ,
consequent: Order.pay)</code> |
| 7. <code>eventOrder (earlier: Payment.authOKProp ,
later: Shipping.shipOrderProp)</code> |

plying that the rule is abstract and not executable. Such rules are augmented with business logic that binds these values. The OWL-P ontology and example protocols are available at <http://research.csc.ncsu.edu/mas/OWL-P/>.

The following is a purchase protocol. It consists of three protocols introduced above: *Order*, *Payment* and *Shipping*. The composition axioms for the purchase protocol are listed in Table 2.2.1:

In the above example, a brief explanation is given for the composition axioms.

The roleDefinition axiom 1 specifies that the role *customer* of *Purchase* protocol is the combination of three distinct roles: role *buyer* in *Order* protocol, *receiver* in *Shipping* protocol, and *payer* in *Payment* protocol. It means that the *customer* may assume the role of *buyer*, *receiver*, *payer*. Similarly, roleDefinition axiom 2 defines that the *merchant* in *Purchase* is the combination of role *seller*, *sender*, *payee* in *Order*, *Shipping*, and *Payment* protocols, respectively.

The dataFlow axioms 3 and 4 are used to specify data dependency between protocols. In the case of axiom 3, defines that the ordered item ID value as specified in *itemID* in *Order* protocol is transferred to the *item* value in the *Shipping* protocol, which specify the item to be shipped. Axiom 4 is used to specify the amount customer is willing to pay.

Implication axioms 5 and 6 are used to associate dependency between semantics properties. Axiom 5 means that when antecedent property *shipmentProp* is set *Shipping* protocol, it implies that the consequent property *goods* in *Order* protocol is set. In plain English,

it says that when the shipment is sent, it implies that the claim *goods* is valid. Similarly, axiom 6 means that when the antecedent property *authOKProp* in *Payment* protocol is set, it will make the consequent *pay* in *Order* protocol true. In other words, it claims that when payment is authorized, payment is assumed to be complete.

EventOrder axiom 7 defines the dependency between two events. It means that property *authOKProp* in *Payment* protocol should be set before property *shipOrderProp* is set.

Above are the composition constraints for the three individual protocols. They specify the constraints that a composition has to observe. The many possible execution paths are considered valid as long as they respect the composition constraints.

Such flexibility is useful in business, as it opens more venues for concurrency, different ways of enactment. However by introducing such dependency, it may introduce deadlocks, conflicts, contradictions, or violation of business requirement. It is important to be certain that all possible composition are fulfilling the business requirements.

2.2.2 Astrophysics Workflow Example

Scientific workflows are to help scientists better to manage their research process. Here is one workflow that Dr. John M. Blondin in Physics Department of NCSU is working on as part of the Terascale Supernova Initiative (TSI) [4]. It is a typical workflow, in the sense that other scientists are facing similar problems but only with different data, different hosts, and different algorithms.

The goal of this astrophysics workflow is to automate the repetitive tasks involved in running a simulation at a remote supercomputing center, and transferring the resulting large data files to local mass-storage and start visualization software on local cluster to generate 3D animations. Manually, the whole process consists following individual tasks:

1. Log into the supercomputer via SSH, prepare the simulation input files, and submit the job to the queue on the supercomputer. The full job may take several days. The first batch of result files will be ready in about 6 hours. The executable and input files are put in a subdirectory (the working directory) on a large scratch disk on the supercomputer, and a shell script is edited for this particular run. The job is submitted to the queuing system using this shell script, which contains parameters like run time,

maximum memory used, and the number of processors used in the MPI simulation.

2. Periodically, log into the supercomputer to monitor the status of the run, and check the validity of the resulting data. The primary data is written out roughly 400 times during the course of a full simulation (i.e. 400 successive time steps). The data is written to disk as one binary file per processor per time step, containing all the physical variables used in the simulation. The files all have the same 5-letter prefix for identification, and a 4 number suffix associated with the MPI task that generated the file.
3. Once a given time step has been written to disk, a program is run on a single processor to read in each binary file (corresponding to a 2D slice of the full 3D simulation domain) and write its contents into a single large netCDF file for each time step. The raw binary slices are then removed, and the single netCDF file can be archived to permanent storage.
4. Transfer files to storage on the local cluster. The netCDF file is then moved over the WAN to a data cluster at NCSU using BBP. This step can be done concurrently with the simulation, processing each time step as it becomes available.
5. Once the data is on the local cluster, a program is run on all the nodes to slice the data into bricks for analysis and visualization. Each node reads a slice of the data from the large netCDF file on the global file system, and writes out that slice on a local disk using the EnSight Gold format. A master file is written out on the global file system identifying the location of the slices (a server-of-server file).
6. Start the EnSight visualization software to generate 3D animation of the simulation. The visualization software can then load this data quickly (since it is reading off of 22 local disks) and process predetermined visuals (images and animations) using canned scripts. The resulting file will be copied to predetermined directory.
7. Download and play the animation thorough a web browser. Since a full simulation can run for hours, perhaps even days, and transferring all the output files across great distances can take equally long, manually running the simulation and transferring the

files is a tedious and error-prone process. In this scenario, a workflow to automate the process removes the repetitiveness and greatly simplifies the task of running a simulation.

This workflow is typical in scientific computing area, as it involves job submission to the supercomputer, transferring huge data, and do customized data analysis. It is important that we can help modeling and debugging the scientific workflows.

2.3 Model Checking

There are two major methods employed in formal verification: model checking and theorem proving.

Model checking is a technique of exhaustive state space search on finite states. Temporal model checking and automaton are two major approaches. Model checking can generate counterexamples representing error or inconsistency. Its main disadvantage is the state explosion problem. Model checkers are able to handle between 100 and 200 state variables. *NUSMV* [9] and *Spin* [21] are well-known examples of model checkers.

Theorem Proving [13] is a technique that expresses both the system and its desired properties as formulas in mathematical logic. The formal system defines a set of axioms and a set of inference rules. Theorem proving is applied to find a proof of the property from the axioms of the system.

We employ model checking techniques to verify the properties of a business composition. We model each eb service or business service as a Finite State Automaton (FSA). A composition from those services is manifested as communicating FSAs. Thus the composition can be considered as a larger FSA.

The exchange of messages between the FSAs is called a conversation. The conversation is a global sequence of messages that are exchanged among participating services, recorded in the order they are sent and received. For a conversation $w = w_0, w_1, w_2, \dots, w_n$, let w_i denote the i th message in w , and $w^i = w_i, w_{i+1}, w_{i+2}, \dots$ denote the i th suffix of w .

Model checking techniques allow one to specify properties and verify whether these properties will hold with regards to the conversations in the composed FSA.

There are two major ways to specify properties in model checking techniques: the Computation Tree Logic (CTL), a branching-time logic, and the Linear Time Logic (LTL) [14]. LTL is a subset of CTL. LTL is used to specify the properties for the service composition in this dissertation, because LTL is supported by the Spin model checker.

2.3.1 Linear Time Logic

We define the LTL properties on conversations as follows. Given a composition FSA, the set of atomic propositions is defined as the power set of message classes, i.e., $AP = 2^M$. LTL properties are constructed from atomic propositions in AP , logical operators $\wedge, \vee, \neg$, and LTL operators **X**, **G**, **U**, **F**. These temporal operators **X**, **G**, **U**, **F** mean “next”, “globally”, “until”, and “eventually”, respectively.

LTL Semantics An LTL formula can be evaluated over a path (sequence) of truth evaluations corresponding to the evolving state of the system and a position on that path. An LTL formula is satisfied by a path if and only if it is satisfied for position 0 on that path. The semantics for the modal operators is summarized in Table 2.2.

Given LTL formulas ϕ and φ , atomic proposition $\psi \in AP$, and a word $w \in M^\omega$ (let $w = w_0, \dots$), then the semantics of LTL formulae can be defined as following. An atomic proposition is modeled as a net of truth valuations where the given atomic proposition is true.

- $w \models \psi$ iff $w \in \psi$
- $w \models \neg\phi$ iff $w \not\models \phi$
- $w \models \phi \wedge \varphi$ iff $w \models \phi$ and $w \models \varphi$
- $w \models \phi \vee \varphi$ iff $w \models \phi$ or $w \models \varphi$
- $w \models \mathbf{X}\phi$ iff $\|w\| > 1$ and $w_1 \models \phi$

In the next state, ϕ is true.

- $w \models \mathbf{G}\phi$ iff for all $0 \leq i \leq n$, $w_i \models \phi$
 ϕ is globally true on subsequent path.

Table 2.2: Semantics of LTL

Textual	Symbolic	Explanation
$N\phi$	$\circ\phi$	Next: ϕ has to hold at the next state. (X is used synonymously.)
$G\phi$	$\Box\phi$	Globally: ϕ has to hold on the entire subsequent path.
$F\phi$	$\Diamond\phi$	Eventually: ϕ eventually has to hold (somewhere on the subsequent path).
$\phi U \psi$	$\phi U \psi$	Until: ψ holds at the current or a future position, and ϕ has to hold until that position. At that position ϕ does not have to hold any more.

- $w \models \phi U \varphi$ iff there exists $0 \leq j \leq n$, such that $w_j \models \varphi$ and, for all $0 \leq i < j$, $w_i \models \phi$

It states that ϕ is true until φ is eventually true.

- $w \models F\phi$ iff $w \models \text{true} U \phi$

$F\phi$ is equivalent to that it holds until ϕ is true.

We say that a composition S satisfies an LTL formula ϕ , i.e., $S \models \phi$, if and only if for each conversation $w \in C(S)$, $w \models \phi$. A popular tool for verifying LTL properties in Spin, as introduced in Section 2.3.2.

Reduction of LTL to two operators One can reduce to two of those operators since the following is always satisfied:

- $F\phi = \text{true} U \phi$
- $G\phi = \neg F \neg \phi$

2.3.2 Spin and Promela Primer

Spin is a model checker for the verification of distributed systems software [21]. It is freely distributed, and often described as one of the most widely used verification systems. To use Spin, a designer must (manually or mechanically) generate models that capture the

essential elements of a distributed system, and formulate requirements as linear temporal logic (LTL) expressions. When Spin determines that a requirement is violated, it generates a counter-case to demonstrate the violation. The verification model represents an abstraction of a design that contains only those aspects of a system that are relevant to the properties one wishes to verify.

Process Meta Language (Promela) is a modeling language appropriate to model the behavior of processes that run concurrently and exchange data between them. Promela is a high level language that supports description of Process using variables (to specify their state) and conditions (to model the behavior). Several interleaved processes communicate through the message channel. Different types of message channels such as asynchronous and synchronous further add value to the language. The protocol example given at the Spin website clearly explains how to abstract a distributed systems communication protocol using sender and receiver process types.

Promela is the modeling language of Spin. Besides control flow constructs such as conditions and loops, Promela incorporates three basic types of objects:

Process. This is an instance of a *proctype*, and defines a behavior.

Data object. Data objects have two scope levels: global and process local. Promela supports basic data types like byte, bool, int, and short.

Message channel. A channel models the exchange of data between processes. Channels are declared either locally or globally.

The notions of Process, Variables and Message Channels in Promela are the highlights of the language that brings it close to the real-life concurrency challenges.

2.3.3 Verifying Composed Services

Hull et al. summarize the research in electronic services [23]. They used Mealy automata to capture compositions. They observe that I/O signatures in Web Service Description Language (WSDL) [8] and pre-/post-conditions in DARPA Agent Markup Language-Services (DAML-S) [10] deal with individual operation in isolation, and cannot fully describe the sequential restrictions over the course of a service. The problem is due to the

lack of explicit modeling of “state” in services. A Mealy machine is used to capture service behavior.

Foster et al. [15], the service composition defined in Business Process Execution Language (BPEL) [7] were verified against the design specification to confirm the expected results from designer and implementers’ viewpoints. The design is modeled as Message Sequence Chart, and compiled into Finite State Process (FSP) notations. The implementation in BPEL is also converted to a FSP notations. The two notations are then checked against each other for inconsistency. In this work, there is little reference to the possible conflict or inconsistency that arise from the utilization of multiple independent services. The verification only concerns the consistency of designers and implementers in an organization. There is little work in the verification of transactional properties.

Recent work by Xiang Fu et al. [17] [18] converts BPEL specification to a guarded automata with unbounded queue for incoming messages. For verification, the intermediate representation is then translated into the target verification language Promela. The model checker Spin performs the verification. Synchronizability of a service is analyzed. In [25], WSFL workflows were also translated to Promela processes, then together with LTL formulas are fed into Spin for verification. The properties checked include reachability, deadlock freedom, and some protocol-specific properties.

Narayanan and McIlraith [26] consider web services described in DAML-S ontology. DAML-S specifications are formalized using situation calculus. Its operational semantics are defined using 1-safe Petri nets. Verification tasks are then expressed as reachability problems in Petri nets, and means to draw the many result from the research of Petri nets.

Deutsch, Sui, and Vianu [12] focuses on the verification of data-driven web services. They uses branching-time logic CTL and CTL* to capture the properties of a web service to be verified. The interaction of the web services with external user is modeled with first-order logic over relational vocabularies.

Chapter 3

Translating Business Protocols to Promela

This chapter describes how the OWL-P protocol specifications can be mechanically translated into Promela in a stepwise manner. This translation enables the verification of a wide range of temporal properties in OWL-P. Figure 3.1 shows the steps involved in verifying the protocols.

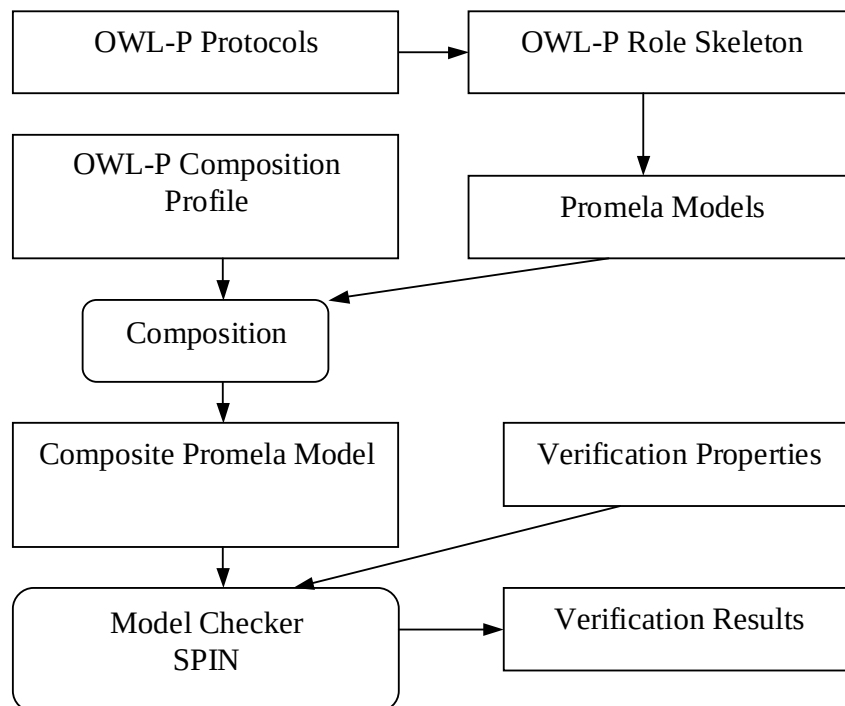


Figure 3.1: The steps of translating OWL-P to Promela and verifying properties

Section 3.1 illustrates how we can translate the role skeletons in OWL-P to Promela. Section 3.2 shows how to translate protocol composition axioms. Section 3.3 how to model commitment systematically.

3.1 Role Skeletons in Promela

This section describes how OWL-P specifications translate to Promela models. Because the Promela notion of process more naturally corresponds to role skeletons than to protocols, we first derive role skeletons from the protocols we wish to verify.

Whereas a protocol specifies the interaction from a global perspective, a role skeleton specifies the protocol from the perspective of a participant. Deriving skeletons for a multiparty protocol can involve some subtlety. Here we assume the algorithm of Desai et al. [11]. For instance, the buyer skeleton in the *Order* protocol would be:

Buyer1. $\text{start} \Rightarrow \text{reqForQuoteS}(\text{?itemID})$

Buyer2. $\text{reqForQuoteP}(\text{?itemID}) \Rightarrow$

$\text{quoteR}(\text{?itemID}, \text{?itemPrice}) \wedge \text{CC}(\text{S}, \text{B}, \text{pay}(\text{?itemPrice}), \text{goods}(\text{?itemID}))$

Buyer3. $\text{quoteP}(\text{?itemID}, \text{?itemPrice}) \wedge \neg \text{rejectQuoteS}(\text{?itemID}, \text{?itemPrice}) \Rightarrow$

$\text{acceptQuoteS}(\text{?itemID}, \text{?itemPrice}) \wedge \text{CC}(\text{B}, \text{S}, \text{goods}(\text{?itemID}), \text{pay}(\text{?itemPrice}))$

Buyer4. $\text{quoteP}(\text{?itemID}, \text{?itemPrice}) \wedge \neg \text{acceptQuoteS}(\text{?itemID}, \text{?itemPrice}) \Rightarrow$

$\text{rejectQuoteS}(\text{?itemID}, \text{?itemPrice}) \wedge \text{CC}(\text{B}, \text{S}, \text{goods}(\text{?itemID}), \text{pay}(\text{?itemPrice}))$

Buyer5. $\text{acceptQuoteP}(\text{?itemID}, \text{?itemPrice}) \Rightarrow \text{end}$

Buyer6. $\text{rejectQuoteP}(\text{?itemID}, \text{?itemPrice}) \Rightarrow \text{end}$

As a convention, a trailing 'S' represents the sending of the specified message and a trailing 'R' represents the receiving of the message. The following are the procedures to map a role skeleton to Promela models. Each ρ_i denotes a role and the a_i are the atoms in the antecedent of a rule.

Translating a role The roles participating in a OWL-P protocol is autonomous. We map each role skeleton to a Promela process. A role receives the messages from a channel designated to it, and will process messages sent to the channel. A nonzero channel length

means that the communication is asynchronous. Each role is by default in its start state.

The mapping is based on the observation that OWL-P is essentially a rule-based language, which will require a rule engine to execute it. Our translation simulates a rule engine through the loop construct. Each OWL-P rule maps to a case in the loop, and has a chance to get executed when the conditions are right.

Listing 3.1: Translating a role skeleton

```
chan to_ρ1 = [99] of {mtype, byte, byte, byte};
proctype ρ1() {
  do
    ...
  od;
}
```

Listing 3.1 shows how a role ρ_1 is translated. Here to_{ρ_1} is a channel of length 99 on which ρ_1 listens. The data type of this channel is a four-tuple consisting of a message type and three bytes corresponding to $-$, $-$, and $-$, respectively. Promela requires a constant for channel buffer size. Here 99 is chosen as it is sufficient for our example.

The process ρ_1 is a loop encoded within a *do* ... *od* Promela loop construct.

Translating messages

For each role, and for each message in the protocol, a Boolean variable records whether the message has already been observed (sent or received) by the role. For each role, for each message in the protocol, and for each parameter of the message, a variable records the value of the parameter known to the role. These global variables allow the verification of the state of the protocol as seen by a role. In Promela, the variables are initialized to zero (false) at declaration.

Listing 3.2 outlines how to mapping the message to code in Promela.

Listing 3.2: Translating messages

```
Message msg1(ρfrom, ρto, slot1, ..., slotn) =>
global: bit ρfrom-msg1 and ρto-msg1;
global: byte ρfrom-msg1-slot1, ..., ρfrom-msg1-slotn;
global: byte ρto-msg1-slot1, ..., ρto-msg1-slotn;
```

In the above translation, OWL-P defines that a message msg_1 is sent from role ρ_{from}

to role ρ_{to} . The message has n data slots. In the translation, we use a global *bit* to note whether the message is sent with variable $\rho_{from_msg_1}$ or received with variable $\rho_{to_msg_1}$.

By parsing the OWL-P message we can extract the variable names for the data holder in the message. They follow the naming convention of *role_message_slot*. When sending the message, the sender role constructs the message first, and the receiving role stores the data received. It is possible to translate the message into a data structure, but they are essentially equivalent.

Mapping of role skeleton rule for sending and receiving

A role skeleton rule governs sending or receiving of messages or the termination of the skeleton in a desirable state. Each rule is mapped to a case statement in the skeleton loop. The antecedent of the rule becomes the condition of the case statement. For simplicity, this dissertation is limited to loop-free protocols. Thus, each rule can fire at most once. This is achieved by having a control variable in the condition of the case statement of each rule. The control variable is disabled after the rule fires. For rules governing the receipt of messages, the condition of the case statement checks whether the message is ready in the channel to be received. Thus, the process makes progress as long as there is a message in the channel.

The mapping is shown in Table 3.1 for sending and for receiving messages. In the mapping for sending message, we use ρ_1 sending rule as the example. The OWL-P rule r_1 means that when $a_1, a_2 \dots, a_n$ are all true, role ρ_1 will send a message from ρ_1 to role ρ_2 . The corresponding Promela code for ρ_1 role skeleton is by adding a case to the *do* loop. It uses a local variable r_1 to make sure that it will be executed only once. Thus the Promela code will include the r_1 as a condition as the condition as well. When $a_1, a_2 \dots, a_n$, and r_1 are all true, the skeleton will enter the code sequence to send a message through a channel called *to- ρ_2* . If there is a conditional commitment involved, we also initiate the creation of the commitment. We then set the tag r_1 to 0, so that the rule will not be executed again. A track variable $\rho_1_msg_1$ is set to signify that the message is sent.

In the case of role ρ_2 receiving message from role ρ_1 , the OWL-P rule r_2 means that when $b_1, b_2 \dots, b_n$ are all true, role ρ_2 will attempt to receive a message from role ρ_1 . The corresponding Promela code for ρ_2 is as shown by adding a case in the *do* loop of the ρ_2

Table 3.1: Mapping of role skeleton rule for ρ_1 sending and ρ_2 receiving message

Rule for ρ_1 sending message: $r_1: a_1 \wedge a_2 \wedge \dots \wedge a_n \Rightarrow$ $msg_1S(\rho_1, \rho_2, \dots) \wedge$ $CC(\rho_1, \rho_2, \dots)$	<pre> proctype $\rho_1()${ bit $r_1=1$; ... do $:: a_1 \ \&\& \dots \ \&\& a_n \ \&\& r_1 \rightarrow$ to-$\rho_2!msg_1(\dots, \rho_1_msg_1_s_i, \dots)$; %create $CC_{id}()$; $r_1=0$; $\rho_1_msg_1=1$; ... od; }</pre>
Rule for ρ_2 receiving message: $r_2: b_1 \wedge b_2 \wedge \dots \wedge b_n \Rightarrow$ $msg_1R(\rho_2, \rho_1, \dots) \wedge$ $CC(\rho_1, \rho_2, \dots)$	<pre> proctype $\rho_2()${ bit $r_2=1$; ... do $:: b_1 \ \&\& \dots \ \&\& b_n \ \&\& r_2 \ \&\&$ to-$\rho_2?[msg_1] \rightarrow$ to-$\rho_2?msg_1(\dots, \rho_2_msg_1_s_i, \dots)$; %receive $CC_{id}()$; $r_2=0$; $\rho_2_msg_1=1$; ... od; }</pre>

Table 3.2: Mapping of role skeleton rule for ρ_1 End

Role skeleton rule for ρ_1 (end): $r_1: a_1 \wedge a_2 \wedge \dots \wedge a_n \Rightarrow \text{end}$	<pre> proctype $\rho_1()${ bit $r_1=1$; ... do :: $a_1 \ \&\& \dots \ \&\& \ a_n \ \&\& \ r_1 \rightarrow$ $r_1=0$; break; ... od; }</pre>
---	--

role skeleton. It uses a local variable r_2 to make sure that it will be executed at most once. Similar to the sending message case, the Promela code includes the r_2 as the condition as well. When $b_1, b_2 \dots, b_n$, and rule r_2 are all true, the skeleton enters the code sequence to receive the message for its channel $to_ \rho_2$. First it checks whether there is message in the channel, otherwise it waits until a message arrives. The tag r_2 is set to 0, so that the rule will not get executed again. A track variable $\rho_1_msg_1$ is set to signify the receipt of the message.

Please note that all variables that appear in the OWL-P rule become variables in the role skeleton.

Mapping of role skeleton end

The termination of the skeleton maps to a *break* statement. The mapping is shown in Table 3.2.

Our translation is simulating a simplistic rule engine executing the rules corresponding to OWL-P rules for relevant roles. Now assume that we have all the rules executed, what will the role skeleton do? It will keep spinning in the do loop. This is important to break the loop once the conditions are met.

In the above mapping, the OWL-P rule is specifying that when condition $a_1, a_2 \dots, a_n$ are true, the role skeleton will exit the do loop. The mapping uses **break** to exit the *do* loop once the condition are met.

3.2 Composite Protocols in Promela

Protocols are rarely used in isolation and must be composed so as to address multiple business goals. For example, purchasing involves a combination of ordering, payment, and shipping. A composite protocol aggregates (component) protocols and is defined by a composition profile. A composition profile stipulates several kinds of composition axioms to specify the various types of dependencies among the component protocols. Table 2.2.1 lists a set of composition axioms for the purchasing example. The notation $axiom_1(property_1:range_1, property_2:range_2 \dots)$ corresponds to the OWL individual $axiom_1$ as the domain of the OWL properties $property_1, property_2 \dots$, and the corresponding ranges $range_1, range_2 \dots$. Each $axiom_i$ belongs to an OWL class corresponding to a schema.

Here we describe the composition axiom schemas and how they apply. Our goal is to construct a Promela model for the composite protocol *Purchase*, given the Promela models of *Order*, *Payment*, and *Shipping* and the set of composition axioms. The effects of the axioms are underlined in the following listings.

Role Definition axioms A role definition axiom defines a role in the composite protocol in terms of the roles of the component protocols. Given the role skeletons for roles ρ_{pi} of component protocols, Listing 3.3 shows how the role ρ_{rd} is defined (P_{rd} is the resultant protocol). Notice how elegant this turns out to be. The defined role skeleton is merely the concurrent execution of the skeletons for the defining roles.

All the roles in the role definition axiom will become an independent process, and will be executed concurrently. In this way, many execution paths are possible, thus enabling the model checker to explore numerous cases for exceptions and errors.

Data Flow Axioms A data flow axiom specifies that a message parameter (originally declared as an external slot) in a component protocol should be assigned to the value of a parameter in another component protocol. Listing 3.4 shows how data flow axioms affect role skeletons. In this Promela translation for data flow axiom, we are using a channel $c_dataFlow1$ to do the data transfer. From the OWL-P data flow axiom, we can parse the roles involved, ρ_{p1} and ρ_{p2} . The axiom specifies that the data in the message field

Listing 3.3: Effect of role definition axioms

```

Role Definition Axiom:
RoleDefinition1 ( define :  $P_{rd}.\rho_{rd}$  ,
unify :  $P_1.\rho_{p1}$  , unify :  $P_2.\rho_{p2}$  , ... ,
 $P_n.\rho_{pn}$  )

```

```

Resultant composite protocol:
proctype  $\rho_{rd}$  () {
    run  $\rho_{p1}()$ ;
    run  $\rho_{p2}()$ ;
    ...
    run  $\rho_{pn}()$ ;
}

```

Listing 3.4: Effect of data flow axioms

```

Axiom:
dataFlow1 ( definition :  $P_1.msg_1.\rho_{p1-msg_1-slot_1}$  ,
usage :  $P_2.msg_2.\rho_{p2-msg_2-slot_2}$  )

```

```

Resultant composite protocol:
chan c_dataFlow1 = [99] of {byte}
proctype  $\rho_{p1}()$  {
    do
        :: ... →
        ...
        c_dataFlow1 !  $\rho_{p1-msg_1-slot_1}$ ;
        ...
    od;
}
proctype  $\rho_{p2}()$  {
    do
        :: ... && c_dataFlow1 ? [ $\rho_{p2-msg_2-slot_2}$ ] →
        c_dataFlow1 ?  $\rho_{p2-msg_2-slot_1}$ ;
        ...
    od;
}

```

$\rho_{p1}\text{-msg1-slot}_1$ should be transferred to role ρ_{p2} message field $\rho_{p2}\text{-msg2-slot}_2$.

It is most important to determine where to place the sending data code, and where the other role skeleton should wait for the data message. We can find the right place by parsing the rules to see when the field is populated in OWL-P rule. The sending code should be placed right after where the field is set. The receiving code should be placed right before the first occasion where the field is referenced.

Event Order Axioms An event order axiom specifies that a proposition in a component protocol should be enabled before a specified proposition in another component protocol. The effect of an event order axiom is similar to that of a data flow axiom. Instead of sending and waiting to receive on a channel, a Boolean flag is set by a role while another role waits for the flag to be set, thus signaling the event. Listing 3.5 shows how event order axioms affect role skeletons. Here a Boolean variable $b_eventOrder1$ is used to signal the event.

In this mapping, the variable $b_eventOrder1$ is initialized to 0. In the earlier role ρ_{p1} , we will need to parse the OWL-P rules to find where it should be set to true. In the later role ρ_{p2} , we also need to determine where the later event should occur. We add $b_eventOrder1$ to the condition formula, so it will be executed only after $b_eventOrder1$ becomes true.

Implication Axioms An implication axiom specifies that a proposition in a component protocol implies a commitment condition or a precondition in another protocol. Listing 3.6 shows how implication axioms affect role skeletons.

In the translation, role ρ_{p1} will have some event that will enable the implication axiom. Role ρ_{p2} will wait until the axiom variable $b_implication1$ is set, then it will enable the consequent variable $cond1$.

When the implication axiom is having consequences that may affect the commitments. It should update commitment translation. For the purchase example, when the shipper ships the goods, it can imply that a global variable $goods$ is fulfilled, thus it needs to update the commitment by the seller to be in the fulfilled state.

Termination Axioms A termination axiom specifies that an event in one protocol may cause the termination of another protocol. For example, if the buyer rejects a quote, the *Payment* and *Shipping* protocols must terminate. This can be effected by appending a case statement conditioned on the event, to all the role processes of the specified protocol.

Listing 3.5: Effect of event order axioms

```

Axiom:
eventOrder1 ( earlier :  $P_1 . \rho_{p1} \text{-} msg_1$  ,
later :  $P_2 . \rho_{p2} \text{-} msg_2$  )

Resultant composite protocol:
bit b_eventOrder1 =0;
proctype  $\rho_{p1}$  () {
    ...
    do
        :: ...  $\rightarrow$ 
        ...
        b_eventOrder1 =1;
        ...
    ...
    od;
}

proctype  $\rho_{p2}$  () {
    ...
    do
        :: ... && b_eventOrder1  $\rightarrow$ 
        ...
    od;
}

```

Listing 3.6: Effect of implication axioms

```

Axiom:
implication1 ( antecedent :  $P_1.\rho_{p1}.msg_1$  , consequent :  $P_2.cond1$  )

Resultant composite protocol:
bit b_implication1 = 0;
bit cond1;
proctype  $\rho_{p1}$  () {
    do
        :: ...  $\rightarrow$ 
        ...
        b_implication1 = 1;
        ...
    od;
}
proctype  $\rho_{p2}$  () {
    do
        :: ...  $\rightarrow$ 
        ...
        b_implication1  $\rightarrow$ 
        cond1 = 1;
        ...
    od;
}

```

3.3 Modeling Commitments in Business Protocols

A commitment is an abstraction of a contract in the business world. It can be used to capture the obligation and rights between the participants in a business transaction.

In a business transaction, there are multiple participants. For example for a general purchase process, there are the buyer, the seller, the shipper, and the payment processor. They form business obligations between them and ensure the successful run of the process. For example, consider when the buyer asks the seller for the price of an item. The seller agrees to sell the item to the buyer if buyer accepts the price. When the seller ships the product, the buyer is expected to pay the correct amount, and the buyer expects to receive the item when he pays. Commitments can capture such obligations quite naturally.

OWL-P captures the commitments between the participants in a business protocol. We need to model those commitments in Promela.

3.3.1 Operations on Commitments

The following operations describe how a commitment c is created, satisfied, and manipulated. The x, y, z are roles participated in a protocol.

1. $CREATE(x, c)$ establishes c , if directly performed by c 's debtor, or because of a conditional commitment.
2. $CANCEL(x, c)$ terminates c as an action of c 's debtor x . Generally, cancellation is constrained and would lead to new commitments.
3. $RELEASE(y, c)$ terminates c by releasing c 's debtor x . This only can be performed by the creditor y .
4. $ASSIGN(y, z, c)$ replaces y with z as c 's creditor.
5. $DELEGATE(x, z, c)$ replaces x with z as the c 's debtor.
6. $DISCHARGE(x, c)$ c is terminated by x fulfilling it.

We need to precisely model the above operations on commitments in Promela.

3.3.2 Translating Commitments into Promela

When attempting to translate commitments into Promela, we encountered the limitation of Promela for modeling all features of commitments. It is preferable that we can associate a conditional expression explicitly and to evaluate the condition at the proper time. This is not doable with Promela. We have to simplify. For example, when we create a conditional commitment, we cannot specify the conditions in an expression. To truly capture a commitment, we need programming languages like C or Java, but modeling in such languages, we will lack the ability to perform model checking on the commitment protocols.

In our earlier work, we modeled each commitment as a process. Due to the difference between commitments in terms of condition variables, each commitment process is unique, making it difficult to provide a more generic approach to model checking. Instead, we now model the commitments as a structure, with its behavior handled by a common Commitment Process (CP).

The Common Commitment Process

The CP maintains an array of commitments defined in Listing 3.7. Each commitment maintains its *creditor* and *debtor* designated by their channels to receive commitment messages. It also maintains a state showing its current status and a slot *pcid* for possible dependency between commitments. For reference, the commitment index in the array is used as ID for identification purposes.

Listing 3.7: Data structure of commitment

```
typedef    Commitment {
    /* Channel to Reach Debtor */
    chan debtor;
    /* Channel to Reach Creditor */
    chan creditor;
    /* store the status of the commitment
       by default it is 0.
    */
    mtype state;
    /* the dependent commitment, by default it is 0 meaning
       no dependency. */
    byte pcid;
}
```

```

/* Commitment array */
Commitment cc[CHANLEN] ;

```

The CP processes the messages concerning commitments from role processes participating in a composition protocol. The CP monitors several channels for messages and updates the corresponding commitment as in Figure 3.2.

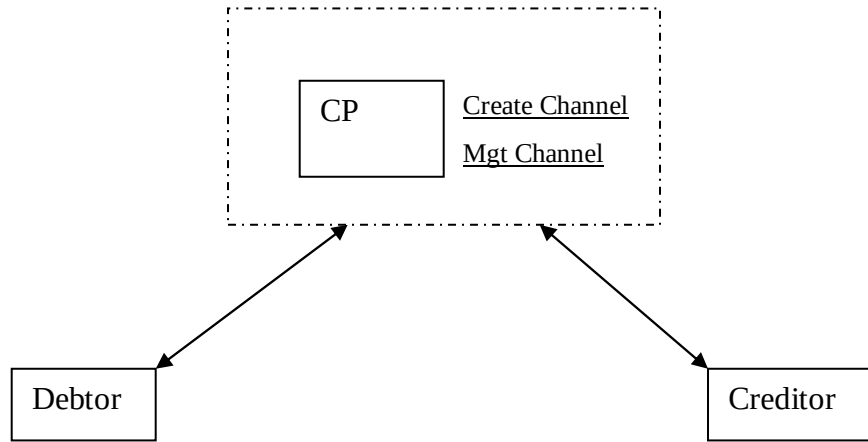


Figure 3.2: The common commitment process

A commitment is created when a *cc_create* message is sent to CP. CP then assigns an empty commitment ID from the commitment array for the commitment and returned to the creator (debtor). The commitment content is forwarded to the other party (creditor) involved in the commitment.

We model the commitment debtor and creditor as channels so as to send messages related with the commitment. CP maintains three channels related to commitment management. The snippet is shown in Listing 3.8.

Listing 3.8: Channels for CP

```

show short c_cnt=1; /* Commitment Counter starts from index 1 */
/* message sent from cc_process */
mtype = { cc_cid };
/* For commitment creation */
chan to_cc_create = [CHANLEN] of { mtype, chan, chan, chan };
/* For update commitment information mtype, chan, cid, state */
chan to_cc_update = [CHANLEN] of { mtype, chan, byte, mtype };
/* For commitment management */
chan to_cc_mgt = [CHANLEN] of { mtype, chan, byte, chan };

```

CP uses the channel *to_cc_create* to receive and create commitments. It uses the channel *to_cc_update* to update the commitment status and the channel *to_cc_mgt* to manage commitments. These channels need to be maintained because Promela can only send fixed format messages on each channel. In reality, one only need to have one communication channel to receive different kinds of messages.

CP monitors its channels for the for following messages.

- *cc_create*(*debtor, creditor, pre_cond, post_cond*) to create commitments. CP will send back the corresponding commitment ID (*cid*).
- *cc_cancel*(*cid*) to cancel the commitment.
- *cc_release*(*cid*) to release the commitment.
- *cc_delegate*(*cid, dchannel*) to delegate to another role process by changing the debtor.
- *cc_assign*(*cid, dchannel*) to assign the commitment to another role by changing the creditor.

It is up to the role process to update CP about the changes to the commitments. CP as a generic commitment process does not understand the specific semantic connotations of a commitment. CP needs to update the commitment structure correctly and changing the states as appropriate.

Modeling Commitment Creation – *CREATE*(*x, c*) Usually, only a debtor role will create a commitment. The following summarizes the operations need to take place on the components in the system.

- Debtor: It creates and sends a *cc_create* message to the CP. It will receive the unique ID from CP.
- CP: The CP chooses an available commitment slot and allocate it to the newly created commitment. It need to store the information related to the commitment. The CP returns the index of the commitment back to the debtor. The CP uses the information in the commitment creation message to update the creditor role and the unique commitment ID.

- Creditor: On receiving the commitment creation message, the creditor needs to record message content, and the unique ID.

Modeling Commitment Cancellation – $CANCEL(x, c)$ Generally, cancellation is constrained and would lead to new commitments. A commitment can only be cancelled by the debtor of the commitment.

- Debtor: It creates and sends a *cc_cancel* message to the CP.
- CP: The CP uses the ID to identify the commitment involved and sets its state to be canceled. The CP also notifies the creditor of the commitment.
- Creditor: On receiving the commitment cancellation message, the creditor understand that the commitment is canceled.

If a new commitment is created because of the cancellation, it is created via the use *cc_create* message.

Modeling Commitment Release – $RELEASE(x, c)$ This can only be performed by the creditor in the commitment.

- Creditor: It releases commitment by sending a *cc_release* message to the CP, with the unique ID.
- CP: The CP uses the ID to identify the commitment involved and sets its state to be released. The CP uses the information in the commitment to notify the debtor.
- Debtor: It records that it is released from fulfilling the commitment.

Modeling Commitment Assignment – $ASSIGN(y, z, c)$ The creditor of the commitment can change the creditor of the commitment.

- Creditor: It assigns the commitment by sending a *cc_assign* message to the CP, with the unique ID and the new creditor name.
- CP: The CP uses the ID to identify the commitment involved and changes the creditor of the commitment. The CP also notifies the debtor on the change of the creditor.

- Debtor: It records the information.

Modeling Commitment Delegation – $DELEGATE(x, z, c)$ The debtor can delegate its obligation to other role that can fulfill the commitment.

- Debtor: It creates and sends a *cc_delegate* message to the CP, along with the commitment id.
- CP: The CP uses the ID to identify the commitment involved and changes the debtor in the commitment. The CP also notifies the creditor of the commitment.
- Creditor: On receiving the commitment delegation message, the creditor understands that the debtor of the commitment is changed.

Modeling Commitment Discharge – $DISCHARGE(x, c)$ The commitment is terminated after it is fulfilled.

- Debtor: It creates and sends a *cc_discharge* message to the CP, along with the commitment id.
- CP: The CP uses the ID to identify the commitment involved and changes its state to be discharged.
- Creditor: On receiving the commitment delegation message, the creditor understands that the commitment is discharged.

3.3.3 Commitment Modeling in Role Process

For each role process involved in a protocol, the commitments must be modeled carefully. It is up to the role processes to update CP with appropriate messages when the situation changes. For example, in the purchasing protocol, CP will not automatically know that shipment of goods has happened and when. So the shipper has to update CP on the commitment associated with the shipment.

Each role process needs to have a channel on which CP can send its messages, and each role process needs to monitor messages coming on this channel.

The implementation of the role process needs to contain the following elements.

- Normal operation: these mainly follow the business process, like the buyer sending a RFQ , and waiting for quote, and sending a purchase order when price is right. Those operations are sequential and event based.
- Monitoring and processing commitment related messages. A role will not know when a commitment might be created, so it will need to poll the channel on which it is receiving commitment messages. This needs to be implemented in a loop.
- Need to maintain multiple commitments: A role process might create new commitments, and participate other commitments created by other roles. For example, the buyer may create a commitment promising to pay if the seller ships. The seller may create a commitment promising to ship if the buyer pays. In this case, the buyer and the seller each participate in two commitments, and need to keep track of the two commitments.

The above requirements make role implementation complex. We need to combine the sequential and looping behavior in one role process. Commonly the polling function can be implemented as a thread in C or Java, but Promela does not have such a feature. However if we implement the role as two processes, one for commitment-related processing and one for sequential processing, we would lose information shared between the two processes as commitment is closely related to the normal business operations.

3.3.4 Commitment Operation Example Scenarios

We use the purchase scenario to give example for commitment operations. The roles seller and buyer are defined in this case. The participants in this example is shown in Figure 3.3.

As the CP is reusable in modeling protocol composition, the individual roles in the protocol need to be modeled according to the translation mechanism. In our purchase protocol example, several global variables are needed in order to verify properties specific to this protocol. Besides the channels, global variables that can be participates in the LTL formula are shown in Listing 3.9.

Commitment Cancellation Example

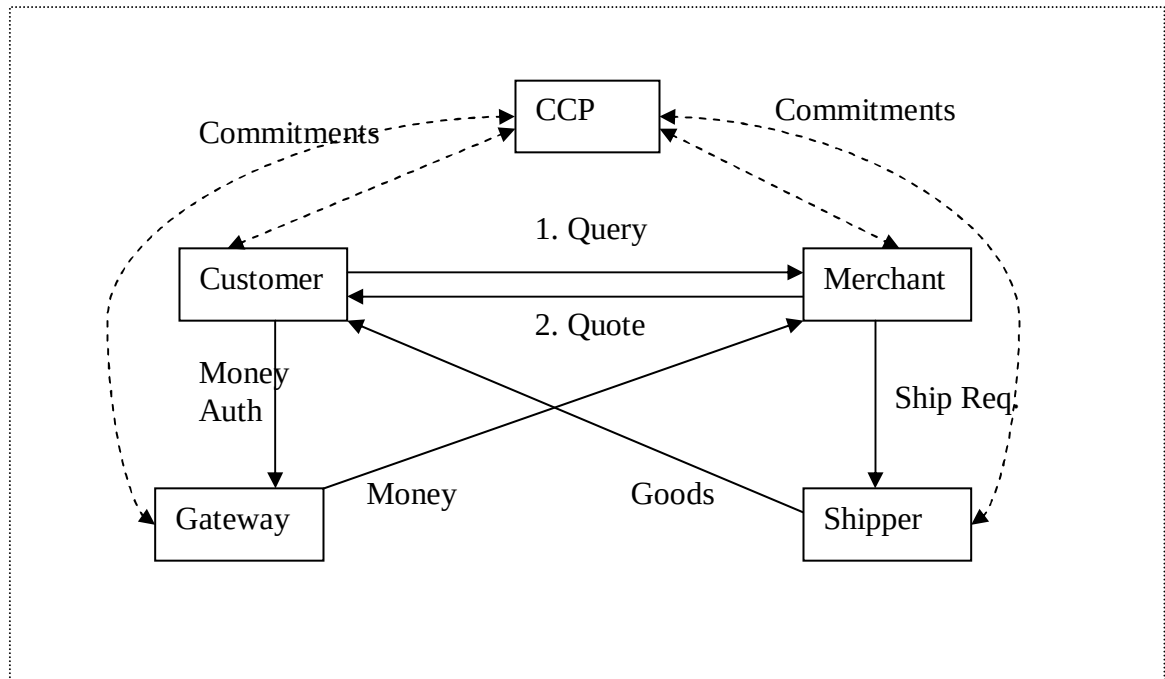


Figure 3.3: Participating roles in the purchase protocol

The buyer asks the seller for the price of an item. The seller replies with a price. The buyer accepts the price and sends a purchase order. Now the buyer creates a commitment that says that if seller ships the product, it will pay the price. However, later the buyer may want to cancel the commitment by paying a penalty.

Commitment Release Example The buyer asks the seller for the price of an item. The seller replies with a price. The buyer accepts the price and sends purchase order. Now the buyer creates a commitment that says that if seller ships the product, it will pay the price.

Listing 3.9: Protocol Specific Global Variables

```

/* Used to Record whether buyer has paid the money*/
bit buyer_paid = 0;
/* Used to record whether seller has received the money*/
bit seller_paid = 0;
/* Used to record whether buyer has received the goods*/
bit b_received = 0;
/* Used to record whether seller has shipped the goods*/
bit s_shipped = 0;

```

The seller also creates a commitment that it will ship if the buyer pays. Now if the buyer wants to cancel the order, it will incur a penalty. The seller can release the commitment, thus forgiving the buyer the penalty.

Commitment Assignment Example The buyer asks the seller for the price of an item. The seller replies with a price. The buyer accepts the price and sends a purchase order. Now buyer creates a commitment that says that if the seller ships the product, it will pay the price. The seller accepts the order, and ships the product. Now the seller can assign receiving the payment from buyer to another entity. The buyer will pay the other party instead.

Commitment Delegation Example The buyer asks the seller for the price of an item. The seller replies with a price. The buyer accepts the price and sends purchase order. Now the buyer creates a commitment that says that if the seller ships the product, it will pay the price. The seller accepts the order, and creates a new commitment saying that if the buyer pays it will ship the product. However, the seller later finds that it cannot fulfill the order. The seller finds another retailer who can fulfill the order for the same price. The seller delegates its obligation to the other retailer.

Commitment Discharge Example The buyer asks the seller for the price of an item. The seller replies with a price. The buyer accepts the price and sends a purchase order. Now the buyer creates a commitment that says that if the seller ships the product, it will pay the price. The seller accepts the order, and creates a new commitment saying that if the buyer pays it will ship the product. Now the seller ships, and the buyer pays. Both commitments are fulfilled now, i.e., discharged.

Commitment Propagation Example It is possible that the fulfillment of one commitment is dependent on the completion of multiple commitments. The following is a 1 to N commitment dependency example. There are four roles A, B, C, and D. Role A promises to pay D five dollars. However role A does not have the money, so he delegates the obligation to his friends B and C. B promises to pay D three dollars and C promises to pay D two dollars. Role A's commitment is (indirectly) fulfilled if B and C pay D for a total of five dollars.

It is also possible that the fulfillment of multiple commitment depends on one com-

mitment. For example, in a real estate example, the commitment of the seller and buyer realtors to receive the commission is dependent upon the commitment of the buyer to buy the house at the agreed price. Such a dependency can be modeled through the creation of conditional commitments.

3.4 Correctness of the Translation

The proposed translation from OWL-P to Promela is modular. Rules within a protocol are mapped into *do ... od* snippets that are executed by each role skeleton. The composition axioms for protocols are also handled modularly. Likewise, commitments and operations on them are handled via the CP. The correctness of the translation can be verified via the inspection, which is aided by the modularity. A rigorous proof would require translating both OWL-P and Promela into the same formalism and would thus face the problem that the other translations were correct.

One way to verify the correctness of OWL-P model and the translated Promela model is through verifying the equivalence of the underlying transition system of two models. Each OWL-P model represents a set of possible execution paths. We need to show that the possible execution paths in Promela model are equivalent.

OWL-P's execution requires a rule engine that understands OWL-P rules and acts upon the rules. Thus the rule engine determines its execution paths. Its execution graph cannot be analyzed without simulating the real execution as the variables used in a rule may be changed, thus enabling other rules.

Our current translation is based on the assumption that all the rules in a protocol will be executed at most once. Otherwise, our translation will require extension. The difficulty mainly lies in determining when to enable or disable a rule. Our translation on the rule skeleton simulates a simplified rule engine, where each rule is translated as a case in switch construct. In this way, each case will get the chance to be executed, when the condition is satisfied for the rule.

Chapter 4

Verification for Business Protocol Properties

We use Spin to check the property and correctness for the business protocol and composition model. We can verify two kinds of properties: general properties such as deadlocks, livelocks, or commitment fulfillment, and protocol-specific properties as discussed in Section 4.3.

4.1 Specifying Property Formulae in Spin

Spin can verify properties specified as LTL formulae. Spin translates LTL formulae into Promela *never claims*. The never claim that is generated from an LTL formula encodes the Buchi acceptance conditions of the formula. A never claim can be used to define system behavior that is of special interest. It is most commonly used to specify behavior that should never happen. The claim is defined as a series of propositions, or Boolean expressions, on the system state. These propositions must become true in the sequence specified for the behavior of interest to be matched. The property variables that can be used in an LTL formula must be public.

Some Generic Properties/Templates are useful when formulating LTL properties:

- Invariance: $\Box\phi$, proposition ϕ is always true. For example, in our purchase example, we want to make sure that for the buyer, he will always receive the goods when he pays as shown in Subsection 4.3.1.

Table 4.1: Some generic formulae of LTL

Symbolically	Explanation
$\Box\phi$	Globally: ϕ has to hold on the entire subsequent path.
$\Diamond\phi$	Eventually: ϕ eventually has to hold (somewhere on the subsequent path).
$\Box\Diamond\phi$	Infinitely often: ϕ will always at some point hold after any point in time. ϕ is recurrently becomes true in the future.
$\Diamond\Box\phi$	Almost always: there is a point in time that ϕ will become true from that time on.

- Response to a stimulus: $\phi \rightarrow \Diamond\psi$, property ϕ becomes true implies that eventually ψ will be true, thus ϕ is a stimulus and ψ is the response to ϕ . For example, in our scenario, we can specify a formula that states if the buyer pays, the seller will eventually ship the product.
- Precedence: $\phi \rightarrow (\psi U \varphi)$, proposition ϕ implies that ψ will be true until φ is true. ϕ is a trigger, or ‘enabling’ condition that determines when this requirement becomes applicable φ is the fulfillment of the requirement, and ψ is a condition that must remain true in the interim.
- Objective: $\phi \rightarrow \Diamond(\psi \vee \varphi)$: It can be read to mean that ϕ is a trigger, or ‘enabling’ condition that determines when the requirement becomes applicable (e.g., the sending of a new data message), then ψ can be the fulfillment of the requirement (e.g., the arrival of the matching acknowledgement), and φ could be a discharging condition that voids the applicability of the check (e.g., an abort condition).

Each of the above generic types of properties can (and will generally have to) be prefixed by temporal operators such as $\Box, \Diamond, \Box\Diamond, \Diamond\Box$. Their semantics is summarized in Table 4.1.

The difference between $\Box\Diamond\phi$ and $\Box\phi$ can be understood as follows. If $\Box\Diamond\phi$ holds, it does not mean that $\Box\phi$ holds at all future points, only that at some future point $\Box\phi$ will hold. But if $\Box\phi$ holds, it automatically mean that $\Box\Diamond\phi$ holds.

4.2 General Properties

The following are some protocol-independent properties that can be expressed in our framework.

4.2.1 Deadlock and Livelock Freedom

Spin verifies deadlock and livelock freedom by default if end states are identified. Using this property we can check whether there are deadlocks in the composition of the protocols. Deadlocks can result from contradictory composition axioms. However, as we have modeled the commitments as processes, if a commitment remains unfulfilled even though all the role processes terminate, Spin would report it as a deadlock. To treat commitment violations separately from role termination, we formulate an LTL property as discussed next.

4.2.2 Termination of Role Processes

We label the termination point for each role process as $term_{\rho}$. If $term_{\rho}$ becomes true, it can never again become false. The termination corresponds to the following LTL formula, where i ranges over the role identifiers:

$$\Diamond \Box \bigwedge_i \rho_i @ term_{\rho_i}$$

This states that it is eventually true that each process ρ_i will be at its termination point labelled with $term_{\rho_i}$.

Using this property we can check whether the business protocol can terminate as expected. We have to note that due to the message passing for communication between role processes, the message received may not be in the order expected, thus potentially resulting in lack of progress.

4.2.3 Commitments in Good States

We model each commitment as a process with a unique name of the form cc_num , where num is a number. The process cc_num has a global variable cc_num_state representing the state of the commitment. All commitment processes do not need to terminate:

it is typical and correct for a conditional commitment process to be in the state *CONDITIONAL* when the protocol computation comes to an end. However, a commitment that ends in the state *BASE* represents an active commitment: termination in a state where some base commitment is active indicates a violation. This property can be formulated as the following LTL formula, where i ranges over commitment names:

$$\Diamond \Box \bigwedge_i cc[i].state \neq BASE$$

This states that eventually all commitment should not be in their *BASE* state. This formula will check for all possibilities that will make the commitment in violation. For example, the ordering of process execution, different order sequence of message being sent, received, and processed, thus making the protocol composition robust.

4.2.4 Unconsumed Messages in Channel

It is important to verify whether all the messages produced are guaranteed to be consumed. For example, in the case of the seller, it may want to verify that when it terminates, there would be no message waiting in its queue. We use the channel polling capability of Promela to check this property. In Promela, $ch?[msg]$ tests whether msg exists in channel ch to be consumed. For a given set of channels ch_i , the above property is captured by the following LTL formula:

$$\Diamond \Box \neg (\bigvee_i ch_i?[msg])$$

This formula captures the property that eventually there should be no message in any Promela message channel.

4.3 Protocol-Specific Properties

Besides the above general properties, Spin can also be used to verify protocol-specific properties. The following properties are based on our example purchasing protocol as described in Section 2.2. Notice that the Boolean variables referred to below are stable in that once they become true, meaning that the corresponding event has occurred, then they remain true. For example, if *gateway_authOK* becomes true, it will stay true.

4.3.1 Ensure Goods if Pay

It is important that the protocol can ensure that after the buyer sends the payment, it should eventually receive a corresponding shipment. The following LTL formula captures this property:

$$\Box(\text{gateway_authOK} \rightarrow \Diamond(\text{shipper_shipOrder} \wedge \text{buyer_acceptQuote_itemID} = \text{receiver_shipment_item}))$$

This states that when money is cleared as signalled by *gateway_authOK*, the seller ships the product represented as *shipper_shipOrder*, and that the item received is the same as the item negotiated in the process.

4.3.2 Ensure Pay if Goods Shipped

From the point view of a seller, he would like to ensure that the buyer will pay the agreed amount, when the product is shipped. The following formula checks that if the shipment is sent, the buyer pays eventually:

$$\Box(\text{shipper_shipOrder} \rightarrow \Diamond(\text{gateway_authOK} \wedge \text{seller_acceptQuote_amount} = \text{payee_captured_amount}))$$

In above formula, variable *shipper_shipOrder* is recording that the seller has shipped the goods, and variable *gateway_authOK* being true means that payment is sent. The above formula states that it is always true that if the seller ships products, it is eventually true that the seller will receive the payment of the agreed upon amount.

4.3.3 No Shipment if Reject

The buyer can choose to accept or reject a quote. However, when he rejects a quote, the goods would not be shipped. The following LTL formula verifies this:

$$\Box \neg (\text{buyer_rejectQuote} \wedge \text{shipper_shipOrder})$$

This states that it is always true that if the buyer rejects a quote, no goods should be shipped.

4.3.4 No Payment if Reject

Similarly, when the buyer rejects a quote, the payment should not be sent. This can be verified by the following LTL formula:

$$\Box \neg (buyer_rejectQuote \wedge gateway_captured)$$

This states that when the buyer rejects a quote, the buyer should not make payments to the seller.

4.4 Verifying Commitment Properties

The interesting commitment properties include the following:

- No Base Level Commitment: We check this property by examining the status of the commitment array. For a commitment to be non-violated, it has to be in either the idle or the discharged state. In order to check the property, we can specify the property as an enumeration of the commitment array, which is not efficient.
- Commitment can only be created by the debtor.
- Commitment can only be canceled by debtor.

4.5 Progressive Refinement

The value of performing protocol modeling before the actual implementation is the possibility of finding inconsistencies in the business protocol and finding scenarios that may jeopardize the smooth and correct running of the business process.

Another value is that we can ascertain that our model satisfies certain business requirements before the implementation, and can discover design errors or omissions. Thus, our approach can help a designer produce a better structured and more robust protocol.

Below we show some example problems that we can discover when modeling the purchase protocol.

Table 4.2: Updated Composition Axioms in Purchase Protocol

- | | |
|----|--|
| 1. | <code>roleDefinition (define: Purchase.customer , unify: Order.buyer ,
unify: Shipping.receiver , unify: Payment.payer)</code> |
| 2. | <code>roleDefinition (define: Purchase.merchant , unify: Order.seller ,
unify: Shipping.sender , unify: Payment.payee)</code> |
| 3. | <code>dataFlow (definition: Order.itemID , usage: Shipping.item)</code> |
| 4. | <code>dataFlow (definition: Order.itemPrice , usage: Payment.amount)</code> |
| 5. | <code>implication (antecedent: Shipping.shipmentProp ,
consequent: Order.goods)</code> |
| 6. | <code>implication (antecedent: Payment.authOKProp ,
consequent: Order.pay)</code> |
| 7. | <code>eventOrder (earlier: Payment.authOKProp ,
later: Shipping.shipOrderProp)</code> |
| 8. | <code>terminate (condition: Order.buyer.rejectQuote ,
Protocol: Payment)</code> |
| 9. | <code>terminate (condition: Order.buyer.rejectQuote ,
Protocol: Shipping)</code> |

4.5.1 Missing Axioms

As we work through the design of a composite protocol, we find that the system stalls when the buyer rejects the quote. After reviewing the counter-example given by Spin, we find that when buyer rejects the quote, the payee and the sender processes are waiting for messages. This motivates us to add new event order axioms or to specify termination axioms (axioms 8 and 9) as in Table 4.5.1. The two terminate axioms will break the loop in the roles of protocol *Payment* and *Shipping*, when the buyer rejects the quote.

4.5.2 Conflicting Axioms

One obvious example that will result in deadlock is when there are conflicting composition axioms. For example, when there is one axiom saying that the buyer will pay only when goods are shipped, and another axiom saying that the seller must receive the payment before shipping goods.

Other problems are more difficult to discern by human, especially when there are many axioms. Hidden conflicts may result from the circular dependencies among various axioms. Those more subtle problems are where verification tools can be most valuable.

Chapter 5

Discussion and Conclusion

This chapter will address the applicability of the work beyond the purchase example, with extensions to other application areas. The limitations of current modeling is discussed in Section 5.2. The literature is reviewed in Section 5.3.

5.1 Applicability

Although we use the *Purchase* process to illustrate the protocol composition and the checking of validity, the overall process can be applied to other business cases. It is also useful in other application areas like scientific workflows where there are interaction protocols involved.

5.1.1 Applicability in Business World

Our approach is applicable in any business process. It will be directly applicable if the interaction is described in OWL-P, as OWL-P is one such language to model interaction protocol. For example, when a car insurance company sells an insurance policy to a customer, it carries the obligation to cover the car when the customer is involved in an accident. For the customer, he needs to pay the insurance premium monthly or every half-year. Here commitments are established between the insurance company and the customer. Also during the purchasing process, the insurance company might sell the policy through a insurance agent, where the commitment is established between the company and customer via the sales agent. Also there maybe negotiation rounds, when the customer is shopping

for the desired coverage, which is similar to the ordering protocol in our purchase example.

When a car accident happens, the customer or the victim will file an insurance claim. The claim will go to a claim department for processing. Some company might outsource the claim process to another company. There is the process of claim validation. When a claim is validated, it may choose an auto mechanic to repair the car. It will pay the mechanic for the service provided.

From the example above, we can see that this scenario may involve many participants with different stakes in the overall process. There is the need for the participants to safeguard their privileges. For example, the mechanic would like to know that he will get the money when fixing the car. There is also a need from the view of customer to get the accident coverage when he pays the premium, no matter what happens between the insurance company and the mechanic.

We can use the procedure outlined in this research to build the model and verify the essential requirements.

5.1.2 Applicability in Scientific Workflow

Currently scientific workflows are still in the early phases of development. In fact, it is a major challenge just to bring each component together to have an executable workflow. To make the workflow able to run correctly, it needs to fulfill the following requirements using the Kepler workflow as an example:

1. Able to access the hosts with correct credentials: This is guaranteed by the user privilege to access those hosts.
2. MoML workflow is well formed. Kepler can automatically check this.
3. The actor will perform as claimed. This is guaranteed by human inspection or testing.
4. The compatibility of links between the actors: Kepler can check requirements.
5. Reliability: the workflow relies on the availability of services used. Redundancy is way to enhance the reliability.

The verification of protocols has limited use in above case, as current priority for scientific workflow is enablement. However, scientific workflows are evolving as follows are:

1. More Service Oriented: The data and tools residing in supercomputer and clusters is becoming more accessible following the Web service paradigm.
2. Well-defined interfaces and semantics: Each service will have a well-defined input and output interfaces, and its externally visible semantics.
3. Autonomous Services: Organization providing private algorithms or research data are independent and may have commercial interests.

The proposed research will be applicable in such a setting. There are two main areas that can benefit from this research. First, current verification models can be applied to services used in the workflow. Those services have a well-defined way of interaction. A scientific workflow will participate in several protocol interactions, and is a result of protocol composition. Thus we can model those interactions in terms of OWL-P, and then we can translate it into Promela for verification purposes. Second, as the invocation of service will involve certain exchange of money or value, we can model those obligations are commitments among the participants, and use the mechanism outlined in this research to verify the design.

5.2 Limitations and Possible Extensions

There are several limitations in our current work, which can potentially be addressed or ameliorated in one way or another through future work.

5.2.1 Expressiveness of Composition Axioms

There are cases where one may want to specify composite formula in the implication axioms or event order axioms. For example, one may want to specify an implication saying that when customer pays by credit card or pays by check, will imply that payment is received. There is a OR formula in this case. This is not yet handled in our implementation.

To support this, the translation mechanism will need to parse the formula and identify the variables involved, which can be arbitrarily complex. In the case of implication axiom, we can identify all the variables used in the antecedent formula. All those variables will need to be defined. We can then define another formula representing the antecedent. In the case of the consequent, we should only allow a conjunction of variables. The consequent will be parsed to generate a number of assignment statements.

5.2.2 Modeling Commitment Propagation

In business scenarios, it is often the case that the cancellation or fulfillment of one commitment can introduce new commitments. For example, when ordering a flight ticket, there is often a clause that penalizes the customer for canceling or changing the ticket. Thus when the customer is canceling the ticket, he can be charged for a penalty such as \$50. To model such behavior, the modeling language should have the ability to dynamically create new commitments based on changing business conditions. Promela is not able to do this due to limited dynamic structure. To overcome the problem, we used a data array of fixed size. When a new commitment is created, we can simply assign one of the available commitment slots to the new commitment.

This limitation complicates modeling of commitment propagation difficult, where the status change of one commitment can affect one or more commitments. We can implement by having a field in commitment structure for pending commitment. However this structure can only model at most 1-to-1 dependencies of the commitments. As Promela does not have the capability of linked lists, it will not be able to model 1-to-n and n-to-1 relationship between the commitments.

A possible extension for future work is to use a finite array to simulate a dynamic data structure. In this way, we will be able to model 1-to-n dependencies between commitments, where n will be fixed ahead of time.

5.2.3 Modeling Repetitive Process

The current OWL-P translation assumes that OWL-P rules will be executed at most once. This makes it difficult to model a seller process where it will be able to serve multiple

negotiation rounds.

To support this, we will need to enable the rules at appropriate time, which can be complex. Any error in enabling the rules at appropriate time will make the model unpredictable. Those requirements should be specified in OWL-P. If available, it should be quite straightforward to map them into Promela.

5.2.4 Supporting Multiple Instances of a Role

Our approach does not model the case where, for example, multiple buyers may want to purchase a product at the same time. In this case, there will be multiple instances of the same role, and may interfere with the correct modeling of the system.

To address this, we can use an ID to group the related role instances into a session. This is similar to how a web server can distinguish two concurrent users. This will introduce more complexity to the translation, as each role process variable will need to be associated with its session ID.

5.3 Literature

Many efforts seek to provide protocol verification by employing the model checking techniques and efforts in the area of standardizing the specification and orchestration of business protocols and web services. They represent the need for reliable and flexible business orchestration platform in the industry.

In open environments, such as the Internet, agents participating in business protocols interact with autonomy and opaque internal design for third party. There is a risk that the participating agents may fail to comply with the given protocol. Venkatraman and Singh [28] present an approach for testing whether the behavior of an agent in open systems complies with a commitment protocol with the commitment protocols specified in temporal logic. Using a formal model of logic-based negotiation, Wooldridge and Parsons [29] define two computational problems: *success* and *guaranteed success*, which loosely correspond to liveness properties. It is important for an agent system to be able to verify the fulfillment or find violation of commitments established during negotiation. Our approach

complements these works by supporting the verification of properties geared toward the composition of commitment-based protocols.

Benatallah *et al.* [6] propose a framework for service protocols modeling, analysis, and management, where protocols are augmented with temporal abstractions. Their focus is on the analysis of compatibility and mutual replaceability of protocols in the face of time constraints in addition to choreographic constraints. They do not consider commitments among the business partners.

Pistore *et al.* [24] work backwards specifying the requirements that the composite service must meet beforehand. They employ automated synthesis techniques to generate the composite service, given a set of component services and composition requirements. Their composition requirements include deadlock freedom and reachability.

Several research efforts study verification of service compositions. These approaches are similar to each other in emphasizing reachability and allied problems. We contrast these collectively with our approach below.

Fu *et al.* [17] support analyzing and verifying interactions among web services. Interactions modeled in BPEL4WS are translated into a formal conversation represented as a guarded finite state automaton (GFSA), which describes some verification properties. The GFSA is then translated to Promela and verified via Spin. Fu *et al.* analyze synchronizability and reachability. Nakajima [25] translates WSFL workflows into Promela to check reachability and deadlock freedom. Foster *et al.* [16], verify a service composition model defined in BPEL against a design specification to confirm the expected results from designers' and implementers' viewpoints. The design is modeled using Message Sequence Charts, and compiled into Finite State Processes (FSP). The implementations in BPEL are also converted to FSPs. The two notations are then checked against each other for inconsistency via model checking.

Narayanan and McIlraith [26] consider web services described in DAML-S. DAML-S specifications are formalized using situation calculus. Its operational semantics are defined using 1-safe Petri nets. Verification tasks are then expressed as reachability problems in Petri nets.

Whereas the above efforts focus on the problems arising from asynchrony such as reachability, realizability, compatibility, and synchronizability, we focus on high-level sub-

ties such as commitment compliance and other protocol-specific properties. Also, our mapping mechanism yields concise and modular Promela models, facilitating the verification of computationally intensive properties. Also, the ability to verify the composition of protocols is novel to our approach.

5.4 Conclusion and Future Directions

We presented a novel way to model business protocols in terms of commitments and the constraints for protocol composition. The correct composition of a business process can be expressed via individual protocol definitions and their composition constraints. Processes can be built from the individual protocols, enabling the checking of large processes.

Our contribution is in providing a generalized mechanism for modeling commitments, and formulating and verifying properties related to commitments. Our results are applicable to a wider range of processes and related protocols, such as scientific discovery processes and workflows. As a result many important properties of business protocols and their compositions can be verified before the actual implementation. Finding problems early can reduce the cost of development, and increase confidence on the validity of the model.

Our future direction is to expand the approach to handle more general scenarios. Another possible direction is to consider more expressive languages for modeling and the properties that can be verified.

Bibliography

- [1] Kepler: An extensible system for scientific workflows. <http://kepler-project.org/>.
- [2] Ptolemy II. <http://ptolemy.eecs.berkeley.edu/ptolemyII/>.
- [3] Scidac/sdm: Scientific data management center. <http://sdm.lbl.gov/sdmcenter/>.
- [4] Shedding new light on exploding stars: Terascale simulations of neutrino driven supernovae and their nucleosynthesis (the terascale supernova initiative): Overview. http://www.scidac.org/HENP/HENP_TSI/reports/TSI2003Annual2.html.
- [5] OWL Web Ontology Language Overview, Feb 2004. <http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
- [6] Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani. On temporal abstractions of web service protocols. In Conference on Advanced Information Systems Engineering Short Paper Proceedings, 2005.
- [7] BPEL. Business process execution language for web services, version 1.1, February 2005. www-106.ibm.com/developerworks/webservices/library/ws-bpel.
- [8] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1, 2001. www.w3.org/TR/wsdl.
- [9] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.
- [10] daml.org. Daml-s: Semantic markup for web services. <http://www.daml.org/services/daml-s/2001/10/daml-s.html>.

- [11] Nirmal Desai, Ashok U. Mallya, Amit K. Chopra, and Munindar P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31(12):1015–1027, 2005.
- [12] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven web services. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 71 – 82, 2004.
- [13] David A Duffy. *Principles of Automated Theorem Proving*. John Wiley & Sons, 1991.
- [14] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. North-Holland, Amsterdam, 1990.
- [15] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *the 18th IEEE International Conference on Automated Software Engineering*, pages 152 – 163, 2003.
- [16] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Tool support for model-based engineering of web service compositions. In *Proceedings of the IEEE International Conference on Web Services*, pages 95 – 102, 2005.
- [17] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting BPEL web services. In *Proceedings of the 13th International World Wide Web Conference*, pages 621 – 630, 2004.
- [18] Xiang Fu, Tevfik Bultan, and Jianwen Su. WSAT: A tool for formal analysis of web services. In *Proceedings of the 16th International Conference on Computer Aided Verification*, pages 510–514, 2004.
- [19] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.
- [20] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, London, 1991.

- [21] Gerard J. Holzmann. The model checker spin. *Software Engineering, IEEE*, 23(5):279–295, 1997.
- [22] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML, May, 2004 (W3C Submission). <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>.
- [23] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. E-services: a look behind the curtain. In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–14, 2003.
- [24] P. Bertoli M. Pistore, P. Traverso and A. Marconi. Automated synthesis of composite BPEL4WS web services. In *Proceedings of the International Conference on Web Services*, 2005.
- [25] Shin Nakajima. Model-checking verification for reliable web service. In *Proceedings of the Workshop on Object-Oriented Web Services*, pages 242 – 251, 2002.
- [26] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. pages 77 – 88, 2002.
- [27] Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
- [28] Mahadevan Venkatraman and Munindar P. Singh. Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, September 1999.
- [29] Michael Wooldridge and Simon Parsons. Languages for negotiation. In W. Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI)*, pages 393–397. John Wiley & Sons, August 2000.