Reasoning Intra-Dependency in Commitments for Robust Scheduling

Mingzhong Wang Kotagiri Ramamohanarao
Dept. of Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia
{minwang, rao}@csse.unimelb.edu.au

Jinjun Chen
Faculty of ICT
Swinburne University of Technology
Victoria 3122, Australia
jchen@swin.edu.au

ABSTRACT

Commitment-modeled protocols enable flexible and robust interactions among agents. However, existing work has focused on features and capabilities of protocols without considering the active role of agents in them. Therefore, in this paper we propose to augment agents with the ability of reasoning about and manipulating their commitments to maximize the system utility. We adopt a bottom-up approach by first investigating the intra-dependency between each commitment's preconditions and result which leads to a novel classification of commitments as well as a formalism to express various types of complex commitment. Within this framework, we provide a set of inference rules to benefit an agent by means of commitment refactoring which enables composition and/or decomposition of its commitments to optimize runtime performance. We also discuss the pros and cons of an agent scheduling and executing its commitments in parallel. We propose a reasoning strategy and an algorithm to minimize possible loss when the commitment is broken and maximize the overall system robustness and performance. Experiments show that concurrent schedules based on the features of commitments can boost the system performance significantly.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems, languages and structures*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability*

General Terms

Design, Reliability, Performance

Keywords

commitment machines, commitment refactoring, agent interaction, robustness, scheduling

1. INTRODUCTION

Agents are autonomous and social entities which cooperate and compete with each other to achieve some goals beyond the limitation of each individual's ability. Accurate and flexible representation of interaction protocols among them is a crucial part of multiagent system design and analysis to ensure coherent and correct execution. [24] argued that the interactions among agents should only constrain their actions to the extent necessary to carry out the given protocol, and no more.

To overcome the inflexibility of traditional approaches which focus on the permissible sequences of the messages in protocols, [4, 23] propose to design agent interactions with the high-level concept of social commitments [2, 19, 8] which represent responsibility from one agent to another. Specifically, different sequences of actions can be derived from the initial state to one of the acceptable termination states with all the constraints imposed by commitments satisfied. If the multi-agent system follows any one of them, the system is considered to be consistent and correct.

However, the issue of how an agent can reason and manipulate its commitments to control and benefit from the provided flexibility has not been addressed. In fact, commitments add an extra dimension of obligation and predictability to agents' behaviors. Therefore, agents can perceive and reason with each other in a more predictive way by generating an optimized but reliable and robust execution plan that can be adjusted at the runtime. In certain situations, agents can convert some sequential execution schedules to be concurrent for better performance while retaining the same correctness and consistency criteria.

To address the effects of commitments on agent's deliberation process, in this paper, we first analyze and classify their inter- and intra-dependency relationships and introduce a formalism to represent and manipulate these relationships. We then define a concurrent scheduling model on the formalism to enable agents to minimize their potential loss when they pursue parallel execution. The main contributions of the paper are the following:

- A novel classification of commitments according to their intra-dependency between the precondition and result.
- A formalism based on the classification for expressing complex commitments.
- The concept and rules of commitment refactoring to help agents compose and decompose their commitments for better runtime performance.
- A concurrent commitment-based scheduling strategy for agents for minimizing possible loss when commitments are broken and thus maximizing the overall system performance. The feature of commitment recoverability in case of breaches is also incorporated into the reasoning and scheduling processes of agents, thus helping to build concurrent and robust multiagent systems.

Cite as: Reasoning Intra-Dependency in Commitments for Robust Scheduling, Mingzhong Wang, Kotagiri Ramamohanarao, Jinjun Chen, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 953–960

The remainder of the paper is organized as follows. Section 2 introduces a motivating example. Section 3 gives the formal definition for complex commitments with various intra-dependency types. Section 4 introduces commitment refactoring, reasoning and manipulating strategies for agents to refine their runtime execution schedules. The design of experiments and results are presented in Section 5. Section 6 provides an overview of the state of the art. Finally, in Section 7 we present our conclusions and perspective for future work.

2. A MOTIVATING EXAMPLE

An interaction protocol defines the permissible sequence of message exchanges. Commitments are applied to represent the obligation between participants of the protocol.

Following [24], a conditional commitment $CC(x, y, \Phi, \Psi)$ denotes an obligation of a debtor x to a creditor y to bringing about Ψ if Φ holds. Here Φ is the precondition and Ψ is the result of the commitment. If Φ is true, it is considered as unconditional and can be denoted as $C(x, y, \Psi)$. After the result Ψ becomes true, the commitment is said to be discharged.

The following Netbill protocol example, borrowed from [14], shows a scenario represented in commitments.

Example 1. *As shown in Figure 1, the interaction begins (s_1) with a customer requesting a quote for some desired goods (s_2), followed by the merchant sending the quote (s_3). If the customer accepts the quote (s_4), then the merchant delivers the goods (s_5) and waits for an electronic payment order (EPO). The goods delivered at this point are encrypted, that is, not usable. After receiving the EPO (s_6), the merchant sends the receipt to the customer (s_7), who can then successfully decrypt and use the goods.*

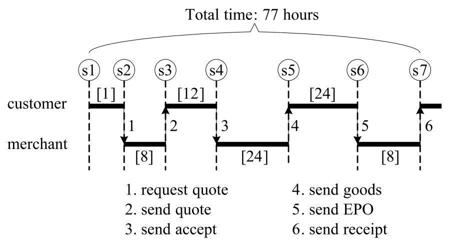


Figure 1: Scenario of purchase protocol

As an addition to the original example, we append an attribute of processing time, represented as *[number of hours]*, for completing each message-related activity. In this way, we can evaluate and compare the efficiency of different schedules. The execution time of the example is computed in a similar way to the estimation generally used in workflow graphs. It will take a total of 77 hours to accomplish the schedule. Table 1 lists the commitments within each state.

A wide range of interaction sequences can be generated and supported from the above commitment protocol by combining various messages in different order when it is allowed [23]. However, the effect of commitment preconditions on the commitment result has not been considered in the model, therefore some order of messages will bring high risk of loss to participating agents, making them implausible. For example, if the merchant in $CC(m, c, accept, goods)$ commences to deliver the goods before the customer agrees with the payment, the merchant risks not receiving any money because the goods delivery completes and discharges the commitment and the customer has no further obligation. Thus, the merchant will tend to choose a defensive approach to wait until they receive the acceptance. In this case, the flexibility provided by the commitment concept is greatly reduced. The resulting state machine may even be condensed into one rigid sequence of messages similar to the traditional methods.

Table 1: Commitments in each state of Interaction Protocol

State	Commitment-related Beliefs
S_1	Null
S_2	*request*
S_3	$CC(m, c, accept, goods) \wedge CC(m, c, payment, receipt)$ where $accept = CC(c, m, goods, payment)$
S_4	$CC(c, m, goods, payment) \wedge C(m, c, goods)$ $\wedge CC(m, c, payment, receipt)$
S_5	$goods \wedge C(c, m, payment) \wedge CC(m, c, payment, receipt)$
S_6	$goods \wedge payment \wedge C(m, c, receipt)$
S_7	$goods \wedge payment \wedge receipt$

We study and classify the possible relations between commitment preconditions and results. With this knowledge, we incorporate the ability of commitment reasoning into the agent's deliberation process to enable rational selection of a beneficial but robust and reliable execution path at run time. We will apply our model to the same example to demonstrate that an agent can actively reason and manipulate its commitments to improve system's robustness as well as performance.

3. INTRA-COMMITMENT DEPENDENCY

This paper's main focus is reasoning about and manipulating the dependency and temporal relationships between agents. We borrow the formalism of the commitment protocol from [4] to represent the commitment and dependency among agents. We also use many temporal logic notions directly to specify the semantics of our extensions to commitments. We note all notions in the paper can be translated into temporal logic [15] and be applied to the general issues on task dependencies. However, we believe the commitment protocol is more natural and less prone to programming errors.

3.1 Commitment Classification

The precondition and result of a commitment have different types of dependencies. Because the condition part is usually carried out by some other agents, the agent can reason about its partners and adjust its behaviors accordingly, and thus benefit from the knowledge of dependency. We formally define the classification of intra-dependency in commitments on top of the interaction protocol which provides a semantic and execution framework for agents and their commitments.

An interaction protocol defines a global state machine incorporating the allowed messages and their occurrence order. The individual agent represents its view of such a protocol as a transition system.

Definition 1. An *agent interaction protocol* is a tuple $\mathcal{P} = (\Pi, \Sigma, s_0, F, M, \Gamma)$ consisting of the following elements:

- $\Pi = u_1, \ldots, u_n$ is a finite set of state variables which are used to represent the interaction related information.
- Σ is a finite set of interaction states. Each state $s \in \Sigma$ is an interpretation of Π, assigning to each variable u in Π a value over its domain.
- $s_0 \in \Sigma$ is the initial state.
- $F \in \Sigma$ is a set of final states.

- M is a finite set of messages.
- Γ is a finite set of transitions ($\Gamma \subseteq \Sigma \times M \times \Sigma$). Each transition $\tau = (s_i, m, s_{i+1}) \in \Gamma$ identifies a source state s_i, a target state s_{i+1} and a message m that is either consumed or produced during this transition.

A computation, or path, of the interaction protocol is defined to be an infinite sequence of states, denoted as $\sigma =< s_0, s_1, \ldots >$. The commitment types are defined by the properties of σ.

Definition 2. **(Commitment types)** A commitment, denoted as $CC(x, y, \Phi, \Psi)$, can be classified according to the temporal constraints between its condition part Φ and result part Ψ. The creditor and debtor of a commitment is omitted when they are clear.

- The commitment is **ordered** if Φ has to be achieved before or at the same time as Ψ is satisfied, formally σ satisfies $\exists i \exists n(i \leq n \wedge s_i \models \Phi \wedge s_n \models \Psi)$. The set of ordered commitments, denoted as OC, which can be further categorized.
  - **strictly ordered**, denoted as $CC(\Phi \sqcap \Psi)$, if Φ has to be satisfied before or at the same time as Ψ and must remain true until Ψ. Formally, σ satisfies $\exists i \exists n(i \leq n \wedge \forall j(i \leq j \leq n \wedge s_j \models \Phi) \wedge s_n \models \Psi)$. The strictly ordered commitment set is denoted as SOC.
  - **weakly ordered**, denoted as $CC(\Phi \sqcup \Psi)$, if $\neg\Phi$ is required between its first occurrence and the completion of the commitment. Formally, σ satisfies $\exists i \exists n(i < n \wedge s_i \models \Phi \wedge s_n \models \Psi \wedge \exists j(i < j \leq n \wedge s_j \models \neg\Phi))$. The set of weakly ordered commitments is denoted as WOC, where $WOC = OC - SOC$.
- The commitment is **unordered** if the occurrence order of Φ and Ψ does not affect the execution or completion of the commitment. The set of unordered commitments is denoted as UC, which can be further categorized into
  - **strictly unordered**, denoted as $CC(\Phi \parallel \Psi)$, if its success requires both Φ and Ψ are satisfied but in any order. Formally, σ satisfies $\exists n(s_n \models \Phi \wedge \Psi)$. The set of strictly unordered commitments is denoted as SUC.
  - **weakly unordered**, denoted as $CC(\Phi \diamond \Psi)$, if regardless of no matter Φ occurs or not, Φ is not satisfied at the time of completion. Formally, σ satisfies $\exists i \exists n(s_n \models \Psi \wedge \forall j(i \leq j \leq n \wedge s_j \models \neg\Phi))$. The set of weakly unordered commitments is denoted as WUC, where $WUC = UC - SUC$.

For example, $CC(m, c, payment, refund)$ is strictly ordered because merchants have to receive money from their customers before they can return it. As another example, for some companies, $CC(m, c, payment, receipt)$ is strictly unordered because they trust customers and allow the payment and receipt issuance to be carried out concurrently. However, others may have a policy of requiring it to be strictly ordered to reduce their risk of money loss. Therefore, the dependency relationship is application-dependent and needs to be defined by users by adding an extra attribute of commitment type to the commitment expression.

The containing relationships among different sets of commitment types can be obtained directly from Definition 2:

$$SOC \subseteq SUC$$
$$OC \subseteq UC$$

As shown in Theorem 1, it can be inferred that WUC is a special case of SOC. However, WUC is listed as a separate category for the completeness of the definition.

Theorem 1. $CC(\neg\Phi \sqcap \Psi) \equiv CC(\Phi \diamond \Psi)$

Proof. The proof is straightforward by expanding the definition of SOC with $\neg\Phi$.

$$\begin{aligned} & CC(\neg\Phi \sqcap \Psi) \\ \Leftrightarrow\ & \exists i \exists n(i \leq n \wedge \forall j(i \leq j \leq n \wedge s_j \models \neg\Phi) \wedge s_n \models \Psi) \\ \Leftrightarrow\ & CC(\Phi \diamond \Psi) \end{aligned}$$ ∎

3.2 Syntax of Commitments

Commitment can be simple in that it contains only atomic proposition as precondition and result. But it is more general to have complex commitments in practice where both condition and result parts can be complex logical formulas with nested commitments. We extend the existing commitment representation to express the complexity caused by various combinations. The operators include the standard connectives, such as $\wedge$, $\vee$ and $\neg$ from propositional logic, and the commitment type operators from Definition 2.

Definition 3. Let p be an atomic proposition defined by a state variable, Φ and Ψ be state formulas. *State-Formulas* in a commitment are defined as:

1. p is a state-formula.
2. $\neg\Phi$, $\Phi \wedge \Psi$ or $\Phi \vee \Psi$ is a state-formula.

Definition 4. Let Φ and Ψ be state-formulas, *Commitment-Formulas* are defined as:

1. $CC(\Phi \sqcap \Psi)$, $CC(\Phi \sqcup \Psi)$, $CC(\Phi \parallel \Psi)$ or $CC(\Phi \diamond \Psi)$ is a commitment-formula.
2. if ϕ, ϕ_1 and ϕ_2 are commitment-formulas, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ or $\neg\phi$ is a commitment-formula.

If we consider a commitment formula $\phi = CC(x, y, c, r)$ as a proposition indicating whether the commitment is held, then ϕ becomes a state formula and nested commitments are supported.

3.3 Semantics of Commitments

The semantics of a commitment are defined by two satisfaction relations (both denoted by $\models$): one for state-formulas and one for commitment-formulas. For the state-formulas, $\models$ is a relation between a protocol $\mathcal{P}$, one of its states s, and a state-formula Φ, denoted as $\mathcal{P}, s \models \Phi$. For the commitment-formulas, $\models$ is a relation between a protocol $\mathcal{P}$, one of its paths σ, and a commitment formula ϕ, denoted as $\mathcal{P}, \sigma \models \Phi$. For convenience, we omit $\mathcal{P}$ if it is clear from the context.

Definition 5. (Semantics of commitments) Let p be an atomic proposition, $\mathcal{P} = (\Pi, \Sigma, s_0, F, M, \Gamma)$ be an interaction protocol, and $s \in \Pi$, Φ, Ψ be state-formulas. The satisfaction relation $\models$ for state-formulas is defined as:

- $s \models p$ if and only if p is true in s
- $s \models \neg\Phi$ if and only if $s \not\models \Phi$
- $s \models \Phi \wedge \Psi$ if and only if $s \models \Phi$ and $s \models \Psi$
- $s \models \Phi \vee \Psi$ if and only if $s \models \Phi$ or $s \models \Psi$

Let ϕ, ϕ_1 and ϕ_2 be commitment-formulas. For path σ the satisfaction relation $\models$ for commitment formulas is defined by:

- $\sigma \models (\phi \equiv CC(\Phi \sqcap \Psi))$ if and only if $\phi \in SOC$
- $\sigma \models (\phi \equiv CC(\Phi \sqcup \Psi))$ if and only if $\phi \in WOC$
- $\sigma \models (\phi \equiv CC(\Phi \parallel \Psi))$ if and only if $\phi \in SUC$
- $\sigma \models (\phi \equiv CC(\Phi \diamond \Psi))$ if and only if $\phi \in WUC$
- $\sigma \models \phi_1 \wedge \phi_2$ if and only if $\sigma \models \phi_1$ and $\sigma \models \phi_2$
- $\sigma \models \phi_1 \vee \phi_2$ if and only if $\sigma \models \phi_1$ or $\sigma \models \phi_2$
- $\sigma \models \neg\phi$ if and only if $\sigma \not\models \phi$

$s_0 \models \neg\Phi \wedge s_0 \models \neg\Psi$ is assumed to simplify the discussion.

The same state-formulas combined by different commitment types may exhibit different properties. In certain conditions, complex formulas can be decomposed into smaller and simpler ones, thus helping to reduce system complexity and improve system concurrency. The following theorem shows the applicable inference rules for commitments.

Theorem 2. *Let Φ, Ψ be state-formulas.* Inference Rules *for commitments are defined as follow:*

- *Distribution over strictly ordered:*
  1. $CC(\Phi \sqcap (\Psi_1 \wedge \Psi_2)) \Rightarrow CC(\Phi \sqcap \Psi_1) \wedge CC(\Phi \sqcap \Psi_2)$
  2. $CC(\Phi \sqcap (\Psi_1 \vee \Psi_2)) \equiv CC(\Phi \sqcap \Psi_1) \vee CC(\Phi \sqcap \Psi_2)$
  3. $CC((\Phi_1 \wedge \Phi_2) \sqcap \Psi) \Rightarrow CC(\Phi_1 \sqcap \Psi) \wedge CC(\Phi_2 \sqcap \Psi)$
  4. $CC((\Phi_1 \vee \Phi_2) \sqcap \Psi) \Leftarrow CC(\Phi_1 \sqcap \Psi) \vee CC(\Phi_2 \sqcap \Psi)$
- *Distribution over weakly ordered:*
  5. $CC(\Phi \sqcup (\Psi_1 \wedge \Psi_2)) \Rightarrow CC(\Phi \sqcup \Psi_1) \wedge CC(\Phi \sqcup \Psi_2)$
  6. $CC(\Phi \sqcup (\Psi_1 \vee \Psi_2)) \equiv CC(\Phi \sqcup \Psi_1) \vee CC(\Phi \sqcup \Psi_2)$
  7. $CC((\Phi_1 \wedge \Phi_2) \sqcup \Psi) \Rightarrow CC(\Phi_1 \sqcup \Psi) \vee CC(\Phi_2 \sqcup \Psi)$
  8. $CC((\Phi_1 \vee \Phi_2) \sqcup \Psi) \Rightarrow CC(\Phi_1 \sqcup \Psi) \wedge CC(\Phi_2 \sqcup \Psi)$
- *Distribution over strictly unordered:*
  9. $CC(\Phi \parallel (\Psi_1 \wedge \Psi_2)) \Rightarrow CC(\Phi \parallel \Psi_1) \wedge CC(\Phi \parallel \Psi_2)$
  10. $CC(\Phi \parallel (\Psi_1 \vee \Psi_2)) \equiv CC(\Phi \parallel \Psi_1) \vee CC(\Phi \parallel \Psi_2)$
  11. $CC((\Phi_1 \wedge \Phi_2) \parallel \Psi) \Rightarrow CC(\Phi_1 \parallel \Psi) \wedge CC(\Phi_2 \parallel \Psi)$
  12. $CC((\Phi_1 \vee \Phi_2) \parallel \Psi) \equiv CC(\Phi_1 \parallel \Psi) \vee CC(\Phi_2 \parallel \Psi)$
- *Distribution over weakly unordered:*
  13. $CC(\Phi \diamond (\Psi_1 \wedge \Psi_2)) \Rightarrow CC(\Phi \diamond \Psi_1) \wedge CC(\Phi \diamond \Psi_2)$
  14. $CC(\Phi \diamond (\Psi_1 \vee \Psi_2)) \equiv CC(\Phi \diamond \Psi_1) \vee CC(\Phi \diamond \Psi_2)$
  15. $CC((\Phi_1 \wedge \Phi_2) \diamond \Psi) \Leftarrow CC(\Phi_1 \diamond \Psi) \vee CC(\Phi_2 \diamond \Psi)$
  16. $CC((\Phi_1 \vee \Phi_2) \diamond \Psi) \Rightarrow CC(\Phi_1 \diamond \Psi) \wedge CC(\Phi_2 \diamond \Psi)$

Proof. Due to space limitations, we only present the proof of Rule 2. Other rules can be proved following the same style.

For Rule 2, Definition 5 states the state-formulas $s_n \models (\Psi_1 \vee \Psi_2) \equiv s_n \models \Psi_1 \vee s_n \models \Psi_2$. Therefore, the commitment-formulas

$$\begin{aligned}
& CC(\Phi \sqcap (\Psi_1 \vee \Psi_2)) \\
\Leftrightarrow\ & \exists i \exists n (i \le n \wedge \forall j (i \le j \le n \wedge s_j \models \Phi) \wedge \\
& (s_n \models \Psi_1 \vee s_n \models \Psi_2)) \\
\Leftrightarrow\ & \exists i \exists n (i \le n \wedge \forall j (i \le j \le n \wedge s_j \models \Phi) \wedge s_n \models \Psi_1) \vee \\
& \exists i \exists n (i \le n \wedge \forall j (i \le j \le n \wedge s_j \models \Phi) \wedge s_n \models \Psi_2) \\
\Leftrightarrow\ & CC(\Phi \sqcap \Psi_1) \vee CC(\Phi \sqcap \Psi_2) \qquad \blacksquare
\end{aligned}$$

4. REASONING AND MANIPULATION OF COMMITMENTS

The knowledge of the compositional structure of a commitment, as well as its dependency type, helps agents to reason about its run-time properties. Therefore, agents can decide to compose, decompose or run in parallel their commitments to find an optimal execution schedule.

4.1 Commitment Refactoring

An agent usually has more than one simultaneous commitment. It can reorganize them for its own benefits as long as it retains the same commitment to others.

Definition 6. Commitment refactoring modifies the commitments of an agent to improve its performance and simplify its structure without changing its external behavior.

There are two main approaches to accomplish commitment refactoring: *composing*, which merges several similar commitments into a complex one; and *decomposing*, which splits one complex commitment into several smaller independent ones. Composing refactoring helps agents to aggregate several related requests together and process them in a batch. This approach benefits agents in the batch processing, such as by the reduction of the cost of resource allocation and expense of locking and unlocking operations. In contrast, decomposing refactoring helps agents to disassemble a batch processing of commitments into several independent processes which can then be executed in parallel.

An agent carries out its commitment refactoring with the inference rules provided in Theorem 2. For equivalent rules $\phi_1 \equiv \phi_2$, either side of the commitment-formula can be substituted by the other one. However, for rules in the form of $\phi_1 \Rightarrow \phi_2$, ϕ_1 is used to substitute ϕ_2 because ϕ_1 has more restrictive constraints on permissible executions. For example, the agent with a commitment $CC((\Phi_1 \vee \Phi_2) \sqcap \Psi)$ can decide to execute either $CC(\Phi_1 \sqcap \Psi)$ or $CC(\Phi_2 \sqcap \Psi)$ by inference from Rule 4.

4.2 Robust Schedule of Single Commitments

Commitment machines can support a wide range of interaction sequences [23]. However, existing research on the commitment protocol has only focused on finding all possible execution sequences without considering the effect of preconditions on the result. In fact, ordered commitments require sequential satisfaction of their conditional and result part to enact the business logic. Although unordered commitments, especially strictly unordered ones, put no constraints on allowed orders, an agent may still choose to retain an order to minimize its possible loss.

If x in $CC(x, y, \Phi \parallel \Psi)$ commences to bring about the result of Ψ before the success of Φ, x may encounter a loss later because satisfaction of Ψ completes and discharges the commitment and y has no further obligation to satisfy Φ. Thus, x will tend to choose a defensive approach to wait until Φ becomes true. Consequently, the flexibility provided by the commitment concept is greatly reduced.

Not only the flexibility of the system, but also the possible performance boost is reduced if agents cannot benefit from carrying out the execution of the conditional and result parts of a commitment concurrently. For example, during the execution of the Netbill protocol, the customer will reach a state holding $CC(c, m, goods \parallel payment)$ in which it faces two choices to continue its execution:

1. waiting until receiving the goods from the merchant before payment, or
2. making the payment directly while waiting for the goods.

The first choice will guarantee that the system always terminates in a proper state in which both goods and payment are true, but will result in a longer processing time (the summation of both the goods delivery time and the payment time). Conversely, the second choice can reduce the execution time greatly (to the maximum value between the goods delivery time and payment time) but leave the customer in risk of losing money because the merchant may choose not to deliver the goods.

We propose to incorporate the ability of commitment reasoning into the agent's deliberation, leading to a new scheduling measure which keeps a balance between the two extremes. As the first step, knowledge of the debtor on a certain strictly-unordered commitment is grouped to evaluate the risk of following the concurrent schedule. The following definitions apply to all types of commitments, but we will focus on discussing strictly unordered ones as others can be inferred intuitively.

Definition 7. A commitment $\phi \equiv CC(x, y, \Phi, \Psi)$ in state s has the property of *Guardianship* if x perceives that there exists an unconditional commitment from any agent A to fulfill the precondition Φ. Formally, $s \vDash \phi \wedge s \vDash (C(A, x, \Phi))$.

Definition 8. A commitment $\phi \equiv CC(x, y, \Phi, \Psi)$ in state s_i has the property of *Recoverability* if x perceives that there exists a compensating commitment $\rho \equiv CC(y, x, \neg\Phi, \Psi')$ in case of commitment breach during the execution which terminates in a state $s_{n'} \vDash \Psi'$. Formally, $\forall j(i < j < n \wedge s_j \vDash \rho)$.

Definition 9. The *Commitment Safety Level* controls the robustness and reliability of a running commitment. According to the properties of *Guardianship* and *Recoverability*, each commitment $\phi \equiv CC(x, y, \Phi, \Psi)$ contained in state s can be classified into four different levels.

0 : *Bare* if neither properties are true.

1 : *Guarded* if only Guardianship is true.

2 : *Recoverable* if only Compensability is true.

3 : *Guaranteed* if both properties are true.

For example, if the customer holds a commitment from the merchant that $C(m, c, goods)$, then $CC(c, m, goods \parallel payment)$ becomes guarded. And if the customer has a method to claim back the payment from a merchant who fails to deliver the goods, the commitment $CC(c, m, goods \parallel payment)$ is considered as recoverable. When it is both guarded and recoverable, the commitment is called guaranteed.

The commitment safety level is especially useful in stating the degree of system consistency in case of commitment breach if the agent decides to execute both the conditional and the result part of a strictly-unordered commitment concurrently. The higher the level of the commitment, the less possible loss in case of exceptions for the agent which schedules its execution in parallel. Based on the safety level of the commitment $CC(x, y, \Phi \parallel \Psi)$, the debtor will reason and plan its next step accordingly:

- If it is bare, it would better choose not to act, otherwise it faces great possible losses
- If it is guarded, it may start to achieve Ψ even though the conditional part has not been satisfied. This decision largely depends on the trustworthy [17] of the agent A who promises to realize Φ.
- If it is recoverable, it can start to achieve Ψ at the beginning as long as the benefit of efficiency outperforms the cost of recovery process.
- If it is guaranteed, it should decide to achieve Ψ in parallel with the agent A for Φ unless efficiency is not an issue of the system.

If the debtor decides to run the commitment which is guarded or guaranteed in parallel mode, it needs to manipulate its beliefs as shown in Algorithm 1. The functions *create*() and *discharge*() represent creation and completion of a commitment respectively and *cancel*() indicates a cancelation [14], while *add*() appends a fact into the beliefbase of the agent. For interaction, "m?" is receiving and "m!" is sending a message. The rules in the algorithm are in the form *head*? | *guard* ← *body* which means when receiving the message *head*, if *guard* is true, then *body* will be executed.

```
require $s_0 \vDash (\phi \equiv CC(x, y, \Phi, \Psi))$, $s_0 \vDash \neg\Phi$, $s_0 \vDash \neg\Psi$,
$s_0 \vDash (\psi \equiv C(A, x, \Phi))$                    /* $A$ is any agent */
/* execute for achieving $\Psi$ */
while $\neg\Psi$ do
    $\Phi? \mid true \leftarrow discharge(\psi), add(\Phi)$ ;
end
$\Psi!$
if $\Phi$ then
    $discharge(\phi)$ ;
else
    /* track progress of $\Phi$ with an additional commitment */
    $create(\theta \equiv CC(x, y, \neg\Phi \wedge \neg\psi \wedge \Psi, \neg\Psi)), \theta!$ ;
    while $\neg\Phi$ do
        /* canceling $\psi$ means no $\Phi$ available, start recovery */
        $cancel(\psi)? \mid \neg\Phi \leftarrow$
                    $cancel(\phi)!, (\neg\Psi)!, discharge(\theta), exit$ ;
        $\Phi? \mid true \leftarrow add(\Phi), discharge(\psi), discharge(\theta)$;
    end
end
```
Algorithm 1: Manipulate commitment for concurrent execution

If θ is triggered, the agent need to achieve a compensating commitment for $\neg\Psi$. For guaranteed commitment, it is already defined as ρ. In other cases, the agent needs to either negotiate with others or seek help from human operators to find such a way.

In the Netbill example, state s_4 contains a guarded commitment $CC(c, m, goods, payment)$. If the customer trusts the merchant's promise of delivering goods, the customer makes the payment directly without waiting for the goods. As a result, the overall interaction time can be reduced from 77 hours in Figure 1 to 53 hours in Figure 2.

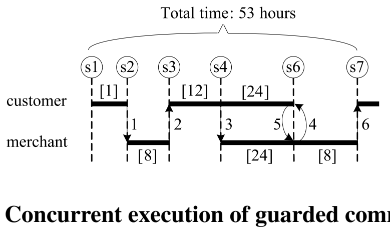


Figure 2: Concurrent execution of guarded commitments

As shown in the example, our model provides a flexible approach to generate concurrent schedules reactively according to the runtime condition by reasoning about the trustworthiness of and promises held by agents to each other. Therefore, even if the program is specified sequentially as in Figure 1, it may be executed in parallel as in Figure 2 by the participating agents. If only recoverable commitments are allowed to be executed in parallel, the concurrent execution can be mapped back into the sequential one for exception recovery.

4.3 Robust Schedule of Combined Commitments

We have discussed the safety level for single commitments in Definition 9. However, the ability of inferring the level of combined commitments is also important for agents to work and operate in more complicated environment. The general way to reason about the safety property of a complex commitment is to decompose it into several child commitments and then aggregate their safety properties together. The result of aggregation affects scheduling and synchronization among its children. The safety level of complex commitment $\phi_1 \wedge \phi_2 \wedge \ldots$ is the minimum safety level of its children and that of $\phi_1 \vee \phi_2 \vee \ldots$ is the maximum one of its children.

For or-combined commitments, it is intuitive for the agent to select the child with the highest safety level, while for and-combined commitments, we define a new parameter to guide the scheduling. Two functions are used in the following definition where $min(S)$ means the minimum value of a set S and $abs()$ returns an absolute value. $min(S) = 0$ if S is empty.

Definition 10. For complex commitment $\phi_1 \wedge \phi_2 \wedge \ldots$, let S_{rec} be the set containing the safety level value of its recoverable children (who are in level 2 or 3) and S_{nrec} be the set containing that of its non-recoverable children (who are in level 0 or 1). The *safety indicator of an and-combined commitment* is equal to $abs(min(S_{rec}) - min(S_{nrec}))$.

By withholding an implicit goal to maintain the safety indicator of its and-combined commitments as high as possible, the agent can schedule its execution more robustly while still providing high parallelism. To keep the safety indicator high, the schedule of the agent will prefer recoverable commitments to non-recoverable ones, and then prefer commitments with lower safety levels within each children group.

The preferences come from the fact that failure of any child leads to overall failure and recovery. Thus, if recoverable children are executed at first, non-recoverable ones have a better chance not to get affected by them. And scheduling the commitment with the lowest level in each group first helps to distribute the possible loss among them because it forces the commitments to execute in a similar progress. Therefore, it is less likely that some commitments will have completed while others have not started yet.

5. EXPERIMENTS AND EVALUATIONS

We have discussed that strictly ordered commitment (*SOC*) requires the debtor and creditor to execute strictly one after another, while strictly unordered commitment (*SUC*) allows them to run concurrently. The Netbill example shows that our method can schedule *SUC*s from a sequence of tasks to run in parallel, thus improving the efficiency of the system. In this section, experiments are performed on applications which can be modelled as directed acyclic graphs (DAGs). As DAG is a fundamental concept to model and represent procedural knowledge [13], our method can also be applied to a wide range of applications domains, such as business process management systems and service-oriented computing [12]. To the best of our knowledge, applying commitment protocol to produce a concurrent schedule for a DAG-modelled task network has not been addressed in previous research. Therefore, we can only provide the evaluation and comparison between the schedule with and without executing strictly unordered commitments in the system concurrently. To simplify the discussion, we only consider the types of *SOC* and *SUC*.

The environment for the experiments was created by adopting the DAG dataset of the Resource-Constrained Project Scheduling Problem, provided in the Project Scheduling Problem Library [16]. Each DAG definition in the dataset specifies a set of task nodes, the precedence relations among them and the duration of each task.

To transform the test sets into our experiments on commitment-based scheduling, we converted each edge in the DAG to be a commitment. For example, the edge from task a to b will be represented as $CC(b, a, Successful(a), Successful(b))$ which means that b promises to complete if a has completed. $Successful(x)$ is a predicate to check if task x has been successfully executed. The execution time of the commitment comes from that of b.

After the conversion, a commitment set which specifies the same semantics as the DAG was obtained. Each element of the set was then randomly assigned to be either an *SOC* or an *SUC*. The parameter $P(SUC)$ was introduced to indicate the probability that a commitment is strictly unordered.

When the feature that *SUC* allows concurrent execution is not considered, all commitments need to preserve the precedence order defined in the DAG as if all of them are *SOC*s. In this case, the total execution time of the commitment set, denoted as $t_{ordered}$, can be computed by finding the critical path [10] of the DAG which is the path with the longest overall duration. On the contrary, if *SUC* is exploited to boost the performance, the concurrent execution time of the commitment set is denoted as $t_{unordered}$. Theoretically, $t_{unordered} <= t_{ordered}$ since the worst case is that no commitment can run concurrently.

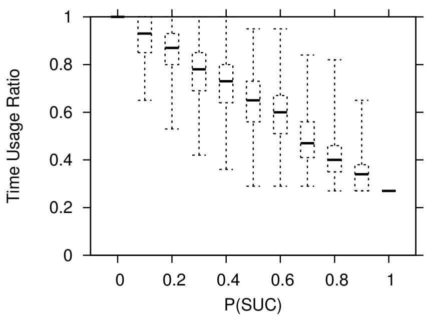
To evaluate the effect of utilizing *SUC*s, we introduced the parameter $TimeUsageRatio = t_{unordered}/t_{ordered}$ which represents the ratio between the duration with and without executing *SUC*s in parallel. The lower the value, the better the performance.

In fact, $t_{ordered}$ should be a fixed value for a certain commitment set while $t_{unordered}$ varies when either $P(SUC)$ or the distribution of *SUC*s in the set changes. Therefore, for each commitment set, $P(SUC)$ was increased from 0 to 1 by 0.1 to show its effect on the performance. As well, for each $P(SUC)$, the experiment was repeated 200 times to find the impact of different distribution of *SUC*s in the commitment set.

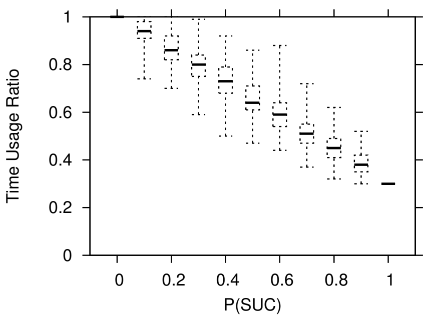
We have experimented on a large number of graph instances from 30-, 60-, 90-, and 120-nodes dateset groups and got similar results. Figure 3 is the box plot showing the randomly selected instances from each group with the name listed in the caption of each sub-graph for clarification.

The results indicate that the system performance has a steady improvement when the percentage of *SUC* increases. The performance improvement is not only reflected on the median value, but also on the the best case and the worst case value. The shrinkage of both the value range and the interquartile range shows that the performance improvement is more stable and obvious at higher $P(SUC)$ value.

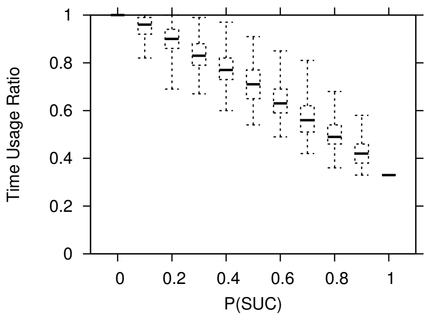
The results also shows that the value range and the interquartile range are wider in 30-nodes cases. Since there are fewer nodes in the graph, the length (in term of the number of edges) of the critical path would also be shorter. Therefore, the distribution of *SUC*s, especially when $P(SUC)$ is low, has stronger impact on the result. In other words, longer path has lower chance to get all its edges



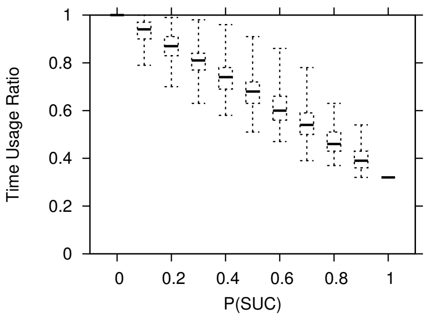
9(a) 30 nodes (j3016_10.sm)



9(b) 60 nodes (j6036_2.sm)



9(c) 90 nodes (j9019_5.sm)



9(d) 120 nodes (j12053_8.sm)

Figure 3: Time usage ratio with variant probability of a commitment being *SUC*

being *SUC*s.

Our approach is proved to be scalable. The increase of graph size does not cause the performance improvement to drop. On the contrary, the boost of performance becomes more accessible for graph instances since the interquartile range keeps on decreasing. Therefore, users can incrementally extend the graph definition and tag eligible commitments to be *SUC*s, thus progressively developing and tuning their systems.

6. RELATED WORK

Commitment theory is one of the core parts of multi-agent research because it helps to specify and constrain the relationship among agents [19, 8]. The formalization and application of commitment protocols have been carried out extensively [22, 21, 14, 23]. Previous work focuses on using commitments to specify the semantics of communication among agents and to reason about all possible execution paths. However, it does not cover the issues of how individual agents can deliberatively reason and manipulate their commitments to make the best selection from the available execution paths or concurrent schedules at run time, especially in the case of possible commitment breaches, which often occur in real applications.

[11] also discusses the over-committed problem in existing research, where agents intend to achieve the condition before the goal of the commitment. They do not provide a balanced reasoning strategy for agents to choose a schedule with the requirements of robustness and concurrency considered from various available execution paths at run time.

[7] formalizes groundedness within an extended BDI (Belief, Desire, Intention) logic, therefore enabling agent to reason about their commitments. However, their focus is on bridging the gap between commitments and the underlying agent framework without considering the detailed features of commitments. Thus, the issues of robustness and concurrency related to commitment fulfilment are not addressed.

[1, 20] model agent execution as goal-plan trees and apply summary information to avoid resource conflicts while pursuing multiple goals in parallel. The work is similar to ours in the sense that agents are active in choosing available execution paths. However, it resides in the abstract level of plan definitions while ours is in social commitments. Therefore, both the purpose and the methods applied in the two research approaches are different.

Leveled-commitment contracting [18], which is similar to the concept of recoverable commitment in our model, is proposed as a backtracking instrument to deal with the uncontrollability of committing and decommitting behaviour of agents, in order to build robust interactions. In fact, it can be utilized in our model to help the design of recovery procedures.

Concurrent scheduling has been heavily studied in high performance computing [6]. However, it remains hard to decompose tasks into parallel parts while preserving complex inter- and intra-dependencies. Commitment-based modeling provides a natural approach to master this complexity by splitting the system into finer components at the semantic level and utilizing the autonomous features of agents to manage concurrent execution. To our knowledge, applying commitments to accompany runtime concurrent scheduling has not been addressed in previous research.

7. DISCUSSION AND CONCLUSIONS

In this paper we have addressed the intra-dependency between the precondition and the result part of a commitment and adopt a bottom-up approach to discuss how agents can benefit from dependency knowledge. An agent can at first benefit from refactoring which composes or decomposes its commitments to suit the runtime situation. With the provided reasoning strategy and corresponding commitment manipulation operations, the agent can evaluate the execution context to make optimized decisions at runtime, thus pursuing a robust and reliable schedule. At the same time, our reasoning and scheduling model enables the agent to follow the concurrent commitment execution to boost the system performance with the minimized possibility of loss, even in case of commitment breaches.

Assume there are n commitments. The complexity of refactoring a single commitment is $O(n)$ because it requires iterating through each commitment. The decision process of reasoning is relatively simple with $O(1)$ for single commitment scheduling and $O(n)$ for combined commitments. In fact, the complexity of reasoning lies in evaluating trustworthiness and recovery cost of an agent. The complexity highly depends on the model used for acquiring these values. In the paper we presume a central reputation server like eBay, and predefined recovery costs. Therefore the complexity is reduced to $O(1)$ as there is only a query process needed. We are currently considering applying research results from trust and process mining to calculate the information automatically.

The experiments performed on DAGs showed that the system performance can be boosted by executing strictly unordered commitments (*SUC*) in parallel. As well, the increase of the percentage of *SUC*s in the commitment set will bring steady improvement. Therefore, users can adopt an incremental approach to specify and tune their task network by progressively tagging commitments as *SUC*s.

To our knowledge this is the first proposal of this kind and has significant practical importance. For example, after dependencies between the preconditions and result of commitments are identified, the possible concurrent executions can be inferred automatically in the form of branching trees. At certain points of the tree, the agent can use inference rules, along with the notion of trust, compensability and recoverability, to select the most beneficial branch. Since this formalism allows agents to make run time decisions, it helps to build robust and recoverable agents.

Our future work concerns three areas. First, recovery is not fully supported for the decomposing refactoring of and-combined commitments. For some unidirectional formulas in Theorem 2, reversing the refactoring is not directly available to recover from failure that happened after the refactoring. Therefore, we hope to record related constraints and information about refactoring into agents' beliefs and provide methods to utilize this knowledge to help with the recovery. Second, we should know the safety level of interaction to determine the robustness of agents' behaviors. We need to develop trust measurements [9] for agents to be able to choose the appropriate level of safety. Finally, we plan to embed the model into existing agent platforms, such as dMARS [5] and 3APL [3], to avoid the burden on programmers of considering the difficult issue of concurrent scheduling management.

8. REFERENCES

[1] B. J. Clement and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *AAAI/IAAI*, pages 495–502, 1999.

[2] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artif. Intell.*, 42(2-3):213–261, 1990.

[3] M. Dastani, M. B. van Riemsdijk, and J.-J. C. Meyer. Programming multi-agent systems in 3apl. In *Multi-Agent Programming*, pages 39–67. 2005.

[4] N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Trans. Software Eng.*, 31(12):1015–1027, 2005.

[5] M. d'Inverno, M. Luck, M. P. Georgeff, D. Kinny, and M. Wooldridge. The dMARS architecture: A specification of the distributed multi-agent reasoning system. *Journal of AAMAS*, 9(1-2):5–53, 2004.

[6] L. Epstein and R. V. Stee. Online scheduling of splittable tasks. *ACM Trans. Algorithms*, 2(1):79–94, 2006.

[7] B. Gaudou, A. Herzig, and D. Longin. Grounding and the expression of belief. In *KR*, pages 221–229, 2006.

[8] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[9] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.

[10] J. J. E. Kelley. Critical-path planning and scheduling: Mathematical basis. *Operations Research*, 9(3):296–320, 1961.

[11] S. M. Khan and Y. Lespérance. On the semantics of conditional commitment. In *AAMAS*, pages 1337–1344, 2006.

[12] S. Kumaran, P. Bishop, T. Chao, P. Dhoolia, P. Jain, R. Jaluka, H. Ludwig, A. Moyer, and A. Nigam. Using a model-driven transformational approach and service-oriented architecture for service delivery management. *IBM Syst. J.*, 46(3):513–529, 2007.

[13] J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-dimensional dynamic knowledge representation. In *LPNMR '01: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 365–378, London, UK, 2001. Springer-Verlag.

[14] A. U. Mallya and M. P. Singh. An algebra for commitment protocols. *Autonomous Agents and Multi-Agent Systems*, 14(2):143–163, 2007.

[15] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[16] Project Scheduling Problem Library - PSPLIB. *http://129.187.106.231/psplib/*, last visited 28 September 2008.

[17] S. D. Ramchurn, D. Huynh, and N. R. Jennings. Trust in multi-agent systems. *Knowl. Eng. Rev.*, 19(1):1–25, 2004.

[18] T. Sandholm and V. R. Lesser. Leveled-commitment contracting: A backtracking instrument for multiagent systems. *AI Magazine*, 23(3):89–100, 2002.

[19] Y. Shoham. Agent-oriented programming. *Artif. Intell.*, 60(1):51–92, 1993.

[20] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting & avoiding interference between goals in intelligent agents. In *IJCAI*, pages 721–726, 2003.

[21] M. Verdicchio and M. Colombetti. A logical model of social commitment for agent communication. In *AAMAS*, pages 528–535, 2003.

[22] D. N. Walton and E. C. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New-York Press, NY, 1995.

[23] M. Winikoff. Implementing commitment-based interactions. In *AAMAS*, pages 868–875, 2007.

[24] P. Yolum and M. P. Singh. Commitment machines. In *ATAL*, pages 235–247, 2001.