

A Model for the Configurable Composition and Synchronization of Complex Trading Activities

Yain–Whar Si, David Edmond, Arthur H.M. ter Hofstede, Marlon Dumas

Centre for Information Technology Innovation
Queensland University of Technology
GPO Box 2434, Brisbane Qld 4001, Australia
{y.si,d.edmond,a.terhofstede,m.dumas}@qut.edu.au

ABSTRACT

With the increasing number of market places and potential trading partners across the e-commerce environment, it will become natural for multiple trading activities to be deployed as part of a single trading strategy. This paper describes a multi-process model for controlling interrelated trading activities. The model includes a powerful generic synchronization construct for building a variety of executable trading engines. The modular design of the construct enables the realization of complex trading schemes, including a number of well known strategies from the financial trading domain. A homogeneous interface is defined to allow seamless integration of control modules built using this construct with elementary trading tasks which directly interact with the trading partners using various negotiation protocols. The model also enables iterative negotiation capabilities, which are essential in any complex trading environment. In addition, the model is designed to be orthogonal to the reasoning model so that any reasoning mechanism can be plugged into it.

Keywords

Complex Trading Activities, Strategies, Negotiation Protocols, Synchronization

1. INTRODUCTION

The rapid development of electronic marketplaces, especially over the Internet, has simultaneously engendered the need and the means to automate trading activities. Online auction houses (eBay, Yahoo), online exchanges (World Chemical Exchange, e-STEEL), and electronic communication networks (Instinet, Island) now provide the basic infrastructure for programmatic product discovery, quote polling, auctioning, bidding, order placement, trade settlement, etc. This has led to the development of software tools that au-

tomate trading tasks. This is essential in order to cope with the explosively growing number of trading opportunities. Already, several tools for trading partners discovery, price tracking, and automated bidding (among others) have emerged.

The next step in this evolution is the automation of *complex* trading activities. These activities are characterized by the need to interact with multiple trading partners and marketplaces concurrently, to trade in multiple units, to comply with temporal constraints, and to deal with specialized knowledge about market mechanisms and domain areas. Because of the subtle interactions between these characteristics, complex trading activities require rigorous planning and control, guided by carefully designed strategies.

Trading strategies can be viewed as a set of guidelines and requirements to perform trading activities. Strategies are essential to every trading situation. They contain detailed plans of how trading activities should be performed. In the case of complex trading activities, trading strategies should capture the interrelationships between concurrent trading tasks potentially involving heterogeneous negotiation protocols. The work reported in this paper addresses these aspects.

Previous studies on the design of strategies for complex trading activities (e.g., [9] [19] [4]) assume a homogeneous negotiation protocol across all trading tasks (e.g., English auction). They are therefore not applicable to activities involving *heterogeneous* negotiation protocols. Our work addresses this heterogeneity by proposing an interface which is used to abstract the internal dynamics of trading tasks. The interface can be applied to any trading task involving a negotiation protocol in which the trader is not required to send proposals. This is the case for example in the Dutch auction protocol, where only the auctioneer makes proposals throughout the negotiation, whereas the trader (i.e., the bidder) only accepts or rejects proposals.

Based on this common interface, a multi-process coordination model is presented, which can be used to configure software modules capable of concurrently participating in multiple negotiations. Using this model, a complex trading activity is carried out by assemblages of elementary trading tasks and synchronization constructs. An elementary trading task handles a negotiation with a given marketplace or trading partner. It acts as a “wrapper”, in the sense that it hides (or rather abstracts from) the specificities of the negotiation protocol imposed by the trading partner. It

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2003 Melbourne, Florida, USA

Copyright 2003 ACM 1-58113-624-2/03/03 ...\$5.00.

also takes local decisions as to whether a given proposal is acceptable or not.

We also define a generic synchronization construct which coordinates a set of trading tasks or other synchronization constructs in a complex trading activity. Instantiations of the construct are also defined in such a way that the trader can dynamically update the constraints during the negotiation, thereby enabling the elementary trading tasks to renegotiate at any time. This dynamic revision capability is essential in any complex trading situation. On the other hand, it adds some complexity to the semantics of the constructs, which must be able to handle multiple types of messages and to process these messages according to the current status of the negotiation. In order to provide a non-ambiguous definition of the construct and elementary trading tasks, formal semantics of them in terms of predicate/transition nets (PrT-nets) are presented.

A brief introduction to the concept of trading task and interrelated trading activities is given in section 2. The homogeneous interface to trading tasks employing various negotiation protocols is presented in section 3. The semantics of a generic synchronization construct is given in section 4. An application example of these synchronization constructs to trading scenarios is given in section 5. In section 6 we briefly review related work before summarizing our ideas in section 7.

2. TRADING TASKS

A key element in specifying trading strategies is the understanding of how basic elements of a trading activity can be defined. An *elementary trading task* is defined as a set of operations required for the purpose of reaching a trading agreement (i.e. a deal). These operations are typically structured in three phases: discovery of trading partners, negotiation, and trade settlement. In this paper, we focus on negotiation, where synchronization between different interrelated trading tasks is required.

An elementary trading task is described by the following attributes: the action to be taken (eg. buy or sell), the description of the item (eg. name, the number of units), the description of the trading partners, and the temporal constraints. An example of an elementary trading task from the financial trading domain is: “Negotiate with seller A to buy 2000 units of BHP with the price of \$18. The trade should be executed before 12:00 13-Nov-2001 and after 10:00 13-Nov-2001.” A *trading activity* may comprise one or more elementary trading tasks. For instance, a buy-sell trading activity may include two elementary trading tasks, one for buying and one for selling. Elementary trading tasks within a trading activity can be interrelated or independent. There are at least two possible types of relationships among two interrelated elementary trading tasks, *complementary* and *alternative*.

Complementary. Two elementary trading tasks are in a complementary [17] relationship when both of them have to be successful or none of them should be successful. For instance, in *Bundle trading* from financial trading, a trader simultaneously purchases or sells an entire portfolio or a cross-section of a portfolio [5]. Index fund managers, index arbitrageurs, hedgers, and equity managers frequently employ bundle trading to maintain the diversification in their portfolio holdings. For example, a fund manager would like to

buy 10000 units of BHP stock and 20000 units of NOKIA stock. In order to re-balance his/her portfolio, the fund manager is determined not to buy any stocks at all if one of these trades is not successful. In other words, all buying tasks should be successful or none of them should be successful. Complementary trading tasks are also widely used in *Options* trading. An option is a right but not the obligation to buy or sell a given security at a certain price within a given time [20]. There are two basic types of options. A *call* option gives the holder the right to buy an asset by a certain date for a certain price and a *put* option gives the holder the right to sell an asset by a certain date for a certain price [12]. A *straddle* is an option trading strategy [12] that involves “simultaneous purchase of an equal number of puts and calls that have the same underlying stock, strike price and expiry month” [20]. The strike price is defined as the predetermined exercise price of a put or call option. Suppose that BHP is currently being traded at \$18, an example of a straddle which includes two complementary trading tasks can be defined as follows:

- Task1: Buy 1000 units of BHP July call option of strike price \$19 at \$0.20 per unit.
- Task2: Buy 1000 units of BHP July put option of strike price \$19 at \$0.12 per unit.

Alternative. Two elementary trading tasks are in an alternative (also known as “substitutive” [17]) relationship when only one of the tasks has to be successful or none of them should be successful. For instance, the manager of a paper production factory may concurrently negotiate with two suppliers for 500 tons of pulp and be planning to choose the one with the lower acceptable offer. In this case, only one of the two trading tasks will be successful. Alternative trading tasks also occur in financial trading. In the US equity market, a significant amount of trades are carried out through one-to-one bargaining between brokers (e.g., in the so-called *upstairs markets* [14]). In this context, a trader can be involved in two alternative tasks in which the same security is sought from two different brokers: these negotiations are synchronized so that at the end, the trader chooses the cheaper offer.

To the best of our knowledge, there is no commercial tool supporting concurrent negotiations involving complementary and alternative trading tasks. Trading activities such as bundle trading and upstairs markets negotiation are still being carried out manually. Furthermore, a single trading activity can involve both complementary and alternative tasks. Suppose that a fund manager is planning a bundle trade which involves simultaneous purchase and sale of several stocks. The plan is:

- buy either 10000 units of BHP from seller S1 or 5000 units of Yahoo from seller S2 and
- sell either 20000 units of NOKIA to buyer B1 or 6000 units of Amazon to buyer B2.

This example suggests that a trading activity can be viewed as a composition of either smaller trading activities or of elementary trading tasks as shown in figure 1. Elementary tasks within the same trading activity can in principle be executed concurrently. However, there are occasions during

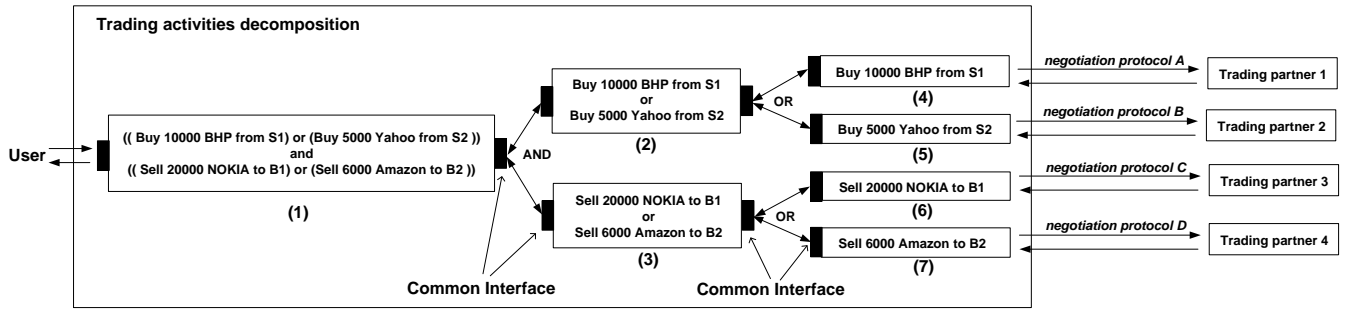


Figure 1: Nested trading activity on bundle trading with heterogeneous negotiation protocols

the concurrent negotiations when the tasks need to be “synchronized” so that a global decision is taken. For example, in figure 1, alternative trading tasks (4) and (5) need to be synchronized to make sure that only one of them will be successful. Alternative trading tasks (6) and (7) also need to be synchronized. In addition, both buying and selling tasks, which are depicted as sub-activities (2) and (3), need to be synchronized so that both of them will be successful or neither will.

The above observations suggest a model where trading tasks are represented as trees in which the leaves denote elementary tasks and the nodes denote *synchronization constructs*. For instance, from the example given in figure 1, the top level activity (node 1) can be replaced with a construct which implements the complementary relationship and two sub-activities (nodes 2 & 3) can be represented with constructs which implement the alternative relationship. Each construct must ensure that its immediate children, regardless of whether they are nodes or leaves, operate according to the requirements set by the semantics of the construct. This delegation of responsibility is propagated level by level from the bottom of the tree till it reaches the root. Before a global decision is taken by the root, it may consult with the user of the software module for a final decision.

Elementary trading tasks may be performed by using different negotiation protocols. This situation is depicted in figure 1. The interactions between the elementary trading tasks and their respective trading partners may vary from one protocol to another since each negotiation protocol has its own distinct characteristics. However, to make synchronization possible, all elementary trading tasks must provide the respective parent nodes with a *common interface* (depicted as filled rectangles in figure 1) which abstracts from the underlying protocols used in the negotiations. Furthermore, due to the hierarchical and composable nature of the model, any components, regardless of whether they are nodes or leaves, should be able to use this common interface for synchronization.

3. HETEROGENEOUS NEGOTIATION PROTOCOLS AND THE COMMON INTERFACE

Trading activities usually involve several parties: a trader will negotiate with his/her counterpart based on some protocol. Negotiation protocols define the circumstances under which the interactions between negotiating parties take place: what deals can be made and what sequences of offers are allowed [16]. These protocols can be classified in

terms of the number of parties participating in the negotiation [16]. In *one-to-one* negotiation, such as bargaining, one trader will negotiate with exactly one other trader. In *one-to-many* negotiation, one trader may negotiate with multiple traders to reach an agreement. Examples of one-to-many negotiations include English, Dutch, Vickrey, and reverse auctions, and as well as tenders. The continuous double auction is an example of a *many-to-many* negotiation in which many traders may negotiate with many other traders.

In the context of negotiation between a trading task and a trading partner (or an auction), we can further classify negotiation protocols into three groups according to following criteria; (a) only the trading task within our model will send proposals, (b) only the trading partner will send proposals, and (c) both parties can send proposals. For instance, in a Dutch auction, only the trading partner (auctioneer) will send proposals whereas in an English auction, the trading task (bidder) should send proposals to the trading partner (auctioneer). Once the proposal is accepted, it must be honoured by the sender. As a result, the negotiation protocols in groups (a) and (c) cannot be used when synchronized decision making is required. Therefore, we will focus on negotiation protocols which do not require the sending of binding proposals.

Trading tasks within the synchronization model may employ different protocols to negotiate with different trading partners. Each negotiation protocol has its own distinct characteristics. For instance, in a Dutch auction, the auctioneer begins with an initial high price and the price descends until someone states a desire to buy at the current price. The sequence of interactions between two participants in a negotiation are different from one protocol to another. Therefore, a generic homogeneous interface is required to invoke, monitor, and control these trading tasks. In addition, within a nested trading activity, an internal node can have other internal nodes or elementary tasks as child-nodes. Due to the hierarchical and composable nature of the model, the same interface should be applied throughout the tree so that an internal node can indistinguishably communicate with other internal nodes and with elementary trading tasks.

In the generic interface, two types of messages are defined: **instructions** (end, negotiate, renegotiate, accept) and **status reports** (ended(e), ready-to-accept(r), unsuccessful(u)). Instructions are always propagated downwards and status reports are propagated upwards. During the negotiation, the user may dynamically revise the constraints of the trading tasks to achieve a better deal. Such revision

may force changes to the status of certain leaf-nodes (e.g. from ready-to-accept to negotiating) and therefore effect the status of their upper level nodes. As a result, additional propagations of instructions and status reports will be required along the tree.

To describe the homogeneous interface and its relation to the inner workings of trading tasks, a predicate/transition net (PrT-net) for a buyer trading task using the Dutch auction protocol is given in figure 2. PrT-nets were first introduced by Genrich and Lautenbach in [8]. They consist of *places*, which are depicted as circles, and *transitions*, which are depicted as rectangles. Edges in PrT-nets may be labelled with linear combinations of tuples consisting of constants or variables. The transitions are annotated with formulas called *transition selectors*, which define the conditions under which the transition can fire. The tokens which flow between places through the transitions can contain data used in the evaluation of the conditions. Interested readers can find more information about PrT-nets in [7].

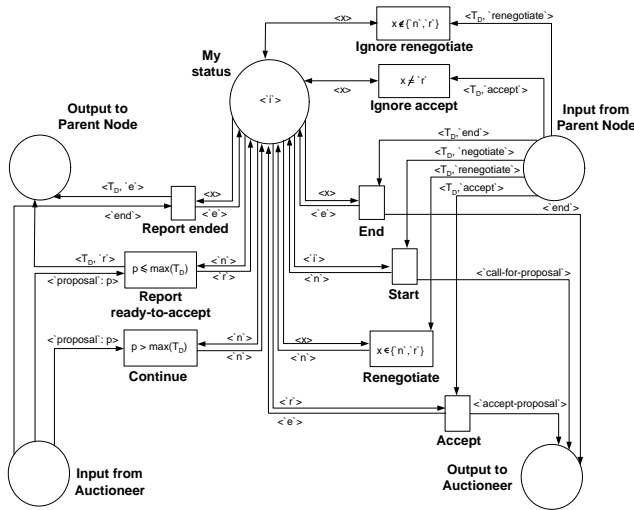


Figure 2: A PrT-net for a trading task T_D with the Dutch auction protocol

The *Input from Parent Node* and *Output to Parent Node* places from figure 2 represent the communication between the trading task and its parent node (or the user). The *Input from Auctioneer* and *Output to Auctioneer* places represent the communication with the trading partner (auctioneer). A token placed in *Output to Auctioneer* indicates that a message is sent to the auctioneer, while a token in the place *Input from Auctioneer* means that a message has been received from the auctioneer. The place *My Status* contains a token corresponding to the current status of the trading task.

Suppose that T_D is the trading task involved. For the purpose of simplification, we assume that there exists a separate message passing mechanism which allows T_D to send and receive price and other related information from the user. We assume that this information is stored in global memory and a function $max(T_D)$ retrieves the most recent price information (e.g. maximum allowable price) for T_D . At any point during a negotiation, T_D can be in one of the following states: idle (i), negotiating (n), ready-to-accept (r), and ended (e). In figure 2, a token $\langle i \rangle$ is placed in *My*

status as an initial marking denoting that T_D is in the idle state. When T_D receives the token containing a ‘negotiate’ command from the parent, it changes its state into negotiating (n) and subsequently sends the ‘call-for-proposal’ to the auctioneer. This situation is depicted by transition *Start*.

In a typical Dutch auction, the auctioneer begins with the initial high price and the price descends until someone states a desire to buy at the current price. The transition *Report ready-to-accept* will be fired if the proposal p from the auctioneer is less than or equal to the maximum price set by the user. As a result, T_D changes its own status from negotiating into ready-to-accept, and informs its parent about the new status. Transition *Continue* is fired when the proposal p sent by the auctioneer is greater than the maximum allowable price. In this case, the trading task maintains its current state and waits for a new proposal from the auctioneer. In some situations, T_D can be outbid by another bidder who might have accepted the proposal from the auctioneer. In this case, the auctioneer sends the ‘end’ message to T_D to signal the end of the negotiation. Upon receiving this message, the transition *Report ended* will be fired and as a result, the state of T_D will be changed into ended and the parent is notified about the failure.

If the current state of T_D is ready-to-accept and a ‘accept’ message has been received from the parent node, the transition *Accept* will be fired and an ‘accept-proposal’ message is sent to the auctioneer. If T_D has received a ‘accept’ message and it is not in the ready-to-accept state, the message should be ignored. By using the transition *Ignore accept*, the ‘accept’ message is consumed without changing the current status of T_D . During the negotiation, the parent node (or the user) may wish to revise the maximum allowable price. If T_D is in negotiating state and a ‘renegotiate’ message is received from the parent node, T_D will continue the negotiation process with the new maximum allowable price without having to change its status. If T_D is in the ready-to-accept state, it changes its state back into negotiating and continues the negotiation process with new constraints. This situation is explicitly modelled in the transition *Renegotiate* in figure 2. If T_D is in a state other than negotiating or ready-to-accept, the ‘renegotiate’ message will simply be ignored by firing transition *Ignore renegotiate*. If the ‘end’ message has been received from the parent node to stop the negotiation, transition *End* is fired and an ‘end’ message is sent to the auctioneer. Subsequently, T_D changes its own state into ‘ended’.

The same common interface can be applied to “abstract” trading tasks with other negotiation protocols such as non-binding bargaining (in which the trading task receives proposals from the trading partner and responds with either “accept” or “reject”).

4. SYNCHRONIZATION

We introduce a generic *synchronization construct* which is simple yet highly adaptable to different trading requirements. Given T_1, \dots, T_m ($m > 0$) trading tasks to be synchronized, the synchronization construct **j OUT OF m** [T_1, \dots, T_m] can be informally defined as “**given m concurrent trading tasks, exactly j OUT OF m tasks are to be successful or none of them should succeed**”. The bundle trading example given in section 2 can be modelled using **j OUT OF m** constructs as depicted in figure 3. The top-level activity and sub-activities from figure 1 are

replaced by appropriate j OUT OF m constructs. By using the common interface, the synchronization constructs and the trading tasks can be structured as a multi-process model. An internal node is responsible for coordinating its immediate child-nodes according to the instructions sent by its parent node (or the user if the node is the root).

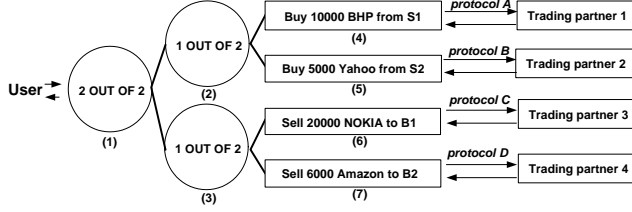


Figure 3: Bundle trading with j OUT OF m synchronization construct

Whenever an internal node receives an instruction from its parent node, it determines which instructions should be sent to its child nodes based on: (a) the semantics of the construct employed at the current node, (b) the instruction received, and (c) the current status of its child nodes. This process is repeated at all internal nodes until it reaches the leaf-nodes (trading tasks). The leaf-nodes in turn report the status of the negotiation to their immediate parent nodes (internal nodes). Based on the status of the child-nodes, an internal node then determines its own status according to the semantics of the construct attached to it, and reports its status to the parent node. This process is repeated at every internal node until it finally reaches the user. Once the user is informed about the current negotiation status, he/she may decide to accept the current deal(s) or to renegotiate with new constraints to try to achieve a better outcome. The propagation of instructions and status reports is repeated for several rounds until the negotiation is ended. Due to the dynamic revision of constraints for renegotiation, the localized decision making, and the composable nature of the model, the generic synchronization construct plays a crucial role and needs to be understood clearly.

In order to provide non-ambiguous definition the j OUT OF m construct, a formal semantics is given in figure 4 using a predicate/transition net. Two places are used to store the status of the current node and the status of child nodes, and four input/output ports are modelled as places. For clarity, the node being addressed is named *self*. The initial marking M_0 of the PrT-net contains

- m tokens $\langle c_1, 'i' \rangle, \dots, \langle c_m, 'i' \rangle$ at the place *My children's status*, where $\langle c_k, 'i' \rangle$ represents the idle state of the k^{th} child node, and m is the total number of child nodes to be synchronized by the current node *self*, and
- a token $\langle 'i' \rangle$ at the place *My status* denoting that the *self* node is idle.

States and message passing. The status of the current node *self* and any child node within the model can be represented as $\langle nd, x \rangle$ where nd is the identification of the node and $x \in \{ 'i', 'n', 'u', 'e', 'r' \}$. At any time, the current node *self* can be in one of the five possible states: (1) i : idle and available for synchronization, (2) n : negotiating, (3)

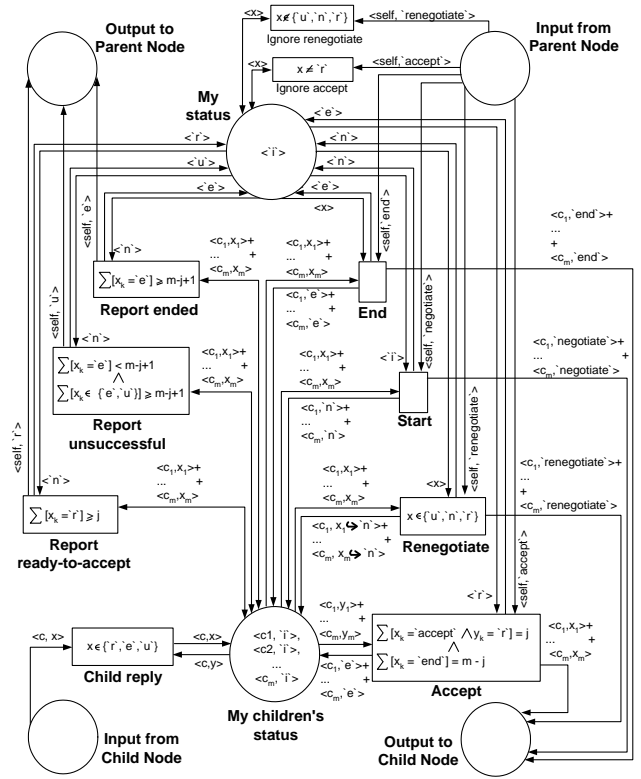


Figure 4: A PrT-net for the j OUT OF m synchronization construct

u : unsuccessful in negotiation but able to renegotiate under new constraints, (4) e : negotiation has ended and unable to renegotiate or continue, and (5) r : ready-to-accept. The possible transitions among these states are depicted in figure 5.

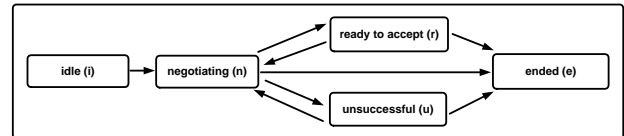


Figure 5: Possible states of a node

After the negotiation process begins, a child node of *self* may reply with three possible states; (1) r : ready-to-accept, (2) u : unsuccessful in negotiation but able to renegotiate under new constraints, and (3) e : negotiation process has ended and unable to renegotiate or continue. These states are stored in the place *My children's status* as individual tokens. The types of token which can be received from the child nodes through the place *Input from Child Node* can be represented as $\langle c_k, x \rangle$ where x is the current state of child c_k and $x \in \{ 'r', 'u', 'e' \}$. The current node *self* may in turn report its own status to the parent node through the place *Output to Parent Node*. The types of tokens which can be sent to the parent node are represented as $\langle self, x \rangle$ where x is the state of the current node *self*, and $x \in \{ 'r', 'u', 'e' \}$. The current state of the node *self* is stored in the place *My status* as a token.

The types of token which can be received from the parent node through the place *Input from Parent Node* can be represented as $\langle self, cmd \rangle$ where *cmd* is the instruction to be carried out by the node *self* and $cmd \in \{ 'end', 'negotiate', 'renegotiate', 'accept' \}$. The meaning of the instructions are:

- *end*: to end all ongoing negotiation processes (below)
- *negotiate*: to begin all negotiation processes (below)
- *renegotiate*: to renegotiate all processes which are in unsuccessful, negotiating or ready-to-accept state
- *accept*: to accept negotiation processes which are ready to accept

Exactly same set of instructions are used when the current node *self* sends the instructions to its child nodes through the place *Output to Child Node*. The types of token which can be sent to the k^{th} child node can be represented as $\langle c_k, cmd \rangle$ where $cmd \in \{ 'end', 'negotiate', 'renegotiate', 'accept' \}$.

Transitions. At the beginning of the negotiation, the current node *self* as well as the child nodes are in the idle state. The transition *Start* is fired when the current node receives the token $\langle self, 'negotiate' \rangle$ from the parent node. After firing, the states of the current node as well as all the child nodes are changed into negotiating. Whenever a child node replies with its current status, it is updated at the place *My children's status* by firing the transition *Child reply*. The transition selector $x \in \{ 'r', 'e', 'u' \}$ ensures that the replies from the child nodes are either ready-to-accept, ended, or unsuccessful.

The transition *Report ready-to-accept* is fired if the current node *self* has enough child nodes in the ready-to-accept state, so that j OUT OF m requirement can be satisfied. This condition is enforced by the transition selector $\sum [x_k = 'r'] \geq j$ which is the simplification of $\sum_{k=1}^m [x_k = 'r'] \geq j$. According to Iverson's convention [13], $[p] = 1$ if p is true or $[p] = 0$ if p is false. Thus, $\sum [x_k = 'r']$ returns the total number of child nodes whose state is ready-to-accept (r).

At any time, if the total number of child nodes in the 'ended' state is greater than or equal to the maximum number of child nodes allowed to be in the 'ended' state ($\sum [x_k = 'e'] \geq m - j + 1$), the transition *Report ended* is fired and a token is sent to the parent node signalling the end of the negotiation. For instance, for a 2 OUT OF 5 construct, if the status of four or more of its child nodes are in the 'ended' state (i.e, they are not able to continue the negotiation), it is certain that the current node will also not be able to continue the negotiation.

The transition *Report unsuccessful* is fired if both of the following conditions hold: (1) the number of child nodes which are in the 'ended' state is still sufficiently low ($\sum [x_k = 'e'] < m - j + 1$) so that it is still possible to continue negotiating, and (2) the total number of child nodes which are in the 'ended' or 'unsuccessful' state is sufficiently high so that it is impossible to achieve j out of m child nodes ($\sum [x_k \in \{ 'e', 'u' \}] \geq m - j + 1$). For instance, if $m = 5$ and $j = 3$, and *My children's status* = $\{ \langle c_1, 'e' \rangle, \langle c_2, 'e' \rangle, \langle c_3, 'u' \rangle, \langle c_4, 'n' \rangle, \langle c_5, 'n' \rangle \}$, the status of a 3 OUT OF 5 construct will be unsuccessful.

During the negotiation, the parent of the current node may send the token $\langle self, 'renegotiate' \rangle$ to instruct the current node *self* to renegotiate with a new set of constraints. The node *self* can renegotiate if it is currently in the unsuccessful, negotiating or ready-to-accept state. In this case, transition *Renegotiate* is fired and a set of tokens, $\langle c_1, 'renegotiate' \rangle$ to $\langle c_m, 'renegotiate' \rangle$ are sent to the child nodes. In addition, the status of child nodes which are in unsuccessful or ready-to-accept are updated to negotiating by the assignment $x_k \leftrightarrow 'n'$ where

$$x_k \leftrightarrow 'n' = \begin{cases} 'n' & \text{if } x_k \in \{ 'u', 'r' \} \\ x_k & \text{otherwise.} \end{cases}$$

If the status of the current node *self* is 'idle' or 'ended', the transition *Ignore renegotiate* is fired so that the token $\langle self, 'renegotiate' \rangle$ is simply consumed without any actions being performed.

The transition *Accept* is fired when the current status of the *self* node is ready-to-accept and it has received the token $\langle self, 'accept' \rangle$ from the parent node. At the time of firing, it is possible that more than j child nodes can be in ready-to-accept state. In the context of current synchronization model, j child nodes will be arbitrarily selected for committing. However, a decision model augmented by a reasoning mechanism, can be used to select those j child nodes with the best overall utility. Due to this separation of synchronization aspects from decision making aspects, our synchronization model can be employed with any decision models which may be developed elsewhere. Upon firing the transition *Accept*, 'accept' messages will be sent to j child nodes which are selected for commitment and 'end' messages will be sent to the remaining $(m - j)$ child nodes. Since the negotiation process is completed, the status of current node *self* as well as the status of all child nodes are updated. If the status of the current node *self* is not in ready-to-accept state and an 'accept' message is received from the parent node, it will be consumed by the transition *Ignore accept*.

If the current node *self* receives the token $\langle self, 'end' \rangle$ from the parent node, the transition *End* is fired and all the child nodes are in turn instructed to stop negotiating. The status of the current node and the child nodes are also changed into 'e' by the transition.

5. EXAMPLE

Suppose that a fund manager is planning to rebalance his/her own portfolio. According to the reports from the research department, the telecommunications sector is likely to outperform the banking sector in the first quarter of the next financial year. The fund manager has decided to rebalance the existing portfolio by selling shares from the banking sector (5000 units of Westpac and 1000 units of National) and buying the shares from telecommunication sector (1000 units of NOKIA and 2000 units of Vodafone).

The fund manager is planning to buy 1000 units of NOKIA shares by concurrently negotiating with sellers S1, S2, and S3 to achieve the best price. The fund manager also estimates that the price of the NOKIA shares may fall during the beginning of the next fiscal year because of the intense competition within the sector. To limit the downside risk, the fund manager has decided to buy 1000 units of NOKIA February put options from one of the two sellers (S4 & S5) with a strike price which is the same as the current price of the share. This strategy is called *Long Stock + Long Put*

Strategy [2]. Synchronization constructs numbered 2, 5, and 6 from figure 6 ensure that buying of both NOKIA shares and options are carried out under the given conditions. In a similar fashion, the fund manager is planning to buy 2000 units of Vodafone shares by concurrently negotiating with sellers S6, S7 and S8. However, he/she estimates that the possibility of Vodafone losing ground in the next quarter is less likely compared to the NOKIA stock. To protect from the potential loss, the fund manager has decided to sell the same amount of Vodafone February call options to one of the buyers B1 and B2 with a strike price which is the same as the current price of the share. This strategy is known as *Long Stock + Short Call Strategy* [2]. Synchronization constructs numbered 3, 7, and 8 from figure 6 make sure that both buying of NOKIA shares and selling of call options will be successful under the given conditions.

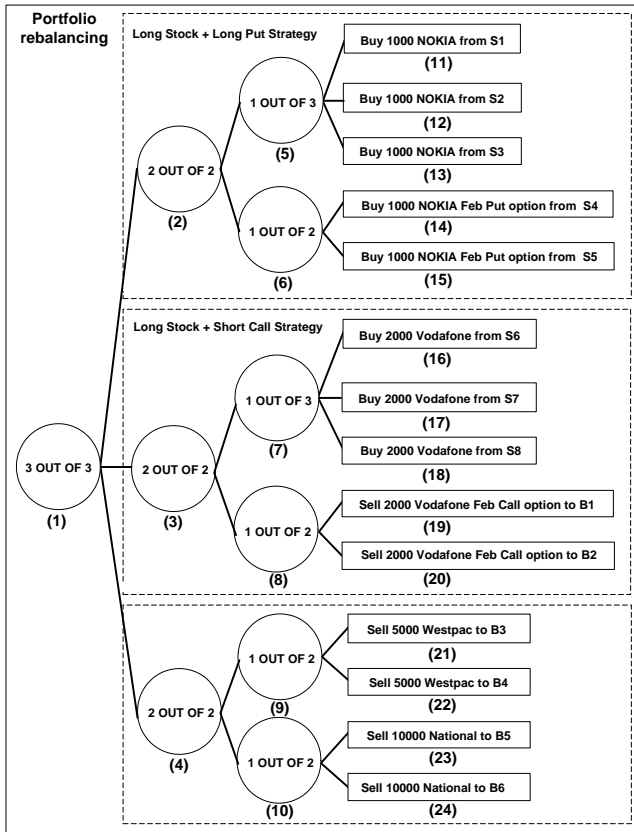


Figure 6: Portfolio rebalancing

To sell 5000 units of Westpac shares, the fund manager is planning to concurrently negotiate with two buyers B3 and B4 and intends to sell to the one with the better (higher) offer. He/she is also planning to sell 10000 units of National shares in a similar way. Synchronization constructs numbered 4, 9 and 10 from figure 6 ensure that both selling tasks will be successful under the given conditions. As part of the portfolio rebalancing, the fund manager is determined not to transact any deals unless all buying and selling requirements are satisfied. Synchronization construct numbered 1 (3 OUT OF 3) from figure 6 ensures that all buying and selling tasks will be successful or none of them will be successful.

6. RELATED WORK

The Michigan Internet AuctionBot [21] is one of the earliest prototypes which allows bidding by human bidders as well as software agents. The AuctionBot is a configurable auction server which manages a large number of simultaneous auctions. In the first Trading Agent Competition (TAC) [11], participant agents bid in simultaneous English auctions for hotel rooms. These hotel rooms have to be packaged with flights and entertainment tickets so as to maximize their utility functions. The scenario from the competition differs from ours, in that the agents in [11] are free to purchase any number of hotel rooms whereas our synchronization construct is designed to achieve an exact number of deals.

Preist et al. [19] have proposed a coordination algorithm for an agent bidding in multiple English auctions for m identical units of an item. The algorithm ensures that the agent makes “at most” m purchases at the end. Their approach differs from ours in at least 3 ways. First, we consider non-binding protocols only and are able to guarantee the “all or none” requirement. Secondly, in [19] there is no mechanism to allow dynamic revision of constraints during the negotiation. Finally, our approach provides modular compositions of complex trading activities.

Anthony et al. [3] have proposed a series of tactics as sets of decision functions for calculating future bids for an agent bidding in multiple heterogeneous auctions for a single item. The agent considered in [3] selects one auction at a time to bid for a single item. In contrast to [3], our approach considers exclusively non-binding negotiation protocols and performs concurrent negotiations to achieve multiple units of the same or different items.

Traditional database transaction processing [10] and advanced transaction models [6] address issues such as consistency, correctness and recovery for transactional tasks. Although the negotiation phase of a trading task is clearly different from a transactional task from the database domain, the multi-process coordination model described in this paper can be seen as an extension of the nested transaction model [18] which allows for the repeated locking of the resources by trading tasks and internal nodes. Although the ready-to-accept status during the negotiation is somewhat similar to placing a lock on a data item in a database, the analogy stops there, since locking a data item does not have the same implications as locking a deal with a trading partner. Unlike the transactional tasks in database domain, rollback or compensating a committed trading task is impractical due to the possibility of financial loss.

To our best knowledge, the only commercial solution that provides a language for specifying trading strategies is TradeStation [1], which is intended for financial applications. EasyLanguage from TradeStation Technologies is a high-level programming language designed for analyzing securities data time-series, implementing trading rules and associated actions. However, it does not provide a direct solution to implement strategies which involve interrelated trading activities. For instance, there are no provisions to express conditional bids (e.g. complementary bids) and synchronization directives.

7. CONCLUSION

This paper described a novel multi-process model for synchronizing interrelated trading activities. A homogeneous

interface is defined to wrap the trading activities involved. In order to specify high-level trading strategies, a generic synchronization construct is introduced, and a formal semantics of the construct is given in PrT-nets. The synchronization construct also provides iterative negotiation capabilities which are essential in any complex trading environment. In addition, the synchronization model is designed to be orthogonal to the reasoning model so that any reasoning mechanisms can be plugged into it.

A proof of concept prototype of the generic synchronization construct and elementary trading tasks were modelled using Renew [15] which is a Java-based high-level Petri net simulator. We have also developed a virtual market place involving several auction servers which can be interfaced with the trading tasks from Renew. Several experiments were performed to simulate the execution of complex trading strategies based on the multi-process model.

8. ACKNOWLEDGMENT

This work is partly funded by the Australian Research Council Grant "Self-describing transactions operating in a large, open, heterogeneous and distributed environment" involving the Queensland University of Technology and GBST Holdings Pty Ltd. The first author is also funded by the University of Macau. The authors wish to thank Nick Russell and Andrew Murdoch for their help and suggestions during the problem formulation.

9. REFERENCES

- [1] Easy Language, TradeStation Technologies. <http://www.tradestationopenplatform.com>.
- [2] Hong Kong Exchanges and Clearing Limited. Common options strategies overview. Available from http://www.hkex.com.hk/TOD/SCORE/english/Part2_0.html, accessed 1-June-2002.
- [3] P. Anthony, W. Hall, V. D. Dang, and N. R. Jennings. Autonomous agents for participating in multiple online auctions. In *Proceedings of IJCAI Workshop on E-Business and the Intelligent Web*, 2001.
- [4] M. Dumas, L. Aldred, G. Governatori, A. H. ter Hofstede, and N. Russell. A probabilistic approach to automated bidding in alternative auctions. In *Proceedings of the 11th International Conference on the World Wide Web (WWW)*, pages 99–108. ACM Press, May 2002.
- [5] P. C. Dunn. Package trading. In W. H. Wagner, editor, *The Complete Guide to Securities Transactions, Enhancing Investment Performance and Controlling Costs*. John Wiley & Sons, 1989. 171–184.
- [6] A. K. Elmagarmid, J. G. M. Yungho Leu, and O. Bukhres. Introduction to advanced transaction models. In A. K. Elmagarmid, editor, *Database transaction models for advanced applications*, pages 33–47. San Mateo, Calif : M. Kaufmann Publishers, 1992.
- [7] H. J. Genrich. Predicate/Transition Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course*, Lecture Notes in Computer Science, Vol. 254, pages 207–247. Springer Verlag, 1987.
- [8] H. J. Genrich and K. Lautenbach. System modelling with high-level Petri Nets. *Theoretical Computer Science, Special Issue on Semantics of Concurrent Computation*, 13:109–136, 1981.
- [9] E. Gimenez-Funes, L. Godo, J. A. Rodriguez-Aguilar, and P. Garcia-Calves. Designing bidding strategies for trading agents in electronic auctions. In Y. Demazeau, editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems, (ICMAS'98)*. IEEE, 1998.
- [10] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc, San Mateo, California, 1993.
- [11] A. Greenwald and P. Stone. Autonomous bidding agents in the trading agent competition. *IEEE Internet Computing*, 5(2):52–60, March–April 2001.
- [12] J. Hull. *Introduction to Futures and Options Markets*. Prentice Hall International, Second Ed., 1995.
- [13] K. E. Iverson. *A Programming Language*. New York : Wiley, 1962.
- [14] D. B. Keim and A. Madhavan. The upstairs market for large-block transactions: Analysis and measurement of price effects. *The Review of Financial Studies*, 9(1):1–36, Spring 1996.
- [15] O. Kummer and F. Wienberg. Renew - the reference net workshop. In *Tool Demonstrations, 21st International Conference on Application and Theory of Petri Nets, Computer Science Department, Aarhus University, Aarhus, Denmark*, pages 28–30, 2000.
- [16] A. R. Lomuscio, M. Woodridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. In F. Dignum and C. Sierra, editors, *Agent-Mediated Electronic Commerce: The European AgentLink Perspective*, LNAI 1991, pages 19–33. Springer Verlag, 2001.
- [17] Y. Matsumoto and S. Fujita. An auction agent for bidding on combinations of items. In *Proceedings of the 5th International Conference on Autonomous Agents, AGENTS'01*, pages 552–559. ACM, 2001.
- [18] J. Moss. Nested transactions and reliable distributed computing. In *Proceedings of the 2nd. Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, PA*, pages 33–39. IEEE CS Press, 1982.
- [19] C. Preist, A. Byde, and C. Bartolini. Economic dynamics of agents in multiple auctions. In *Proc. of the 5th International Conference on Autonomous Agents*, pages 545–551, Montreal, Canada, May 2001. ACM Press.
- [20] C. Tate. *Understanding Options Trading in Australia*. Information Australia, 1990.
- [21] P. R. Wurman, M. P. Wellman, and W. E. Walsh. The Michigan Internet AuctionBot: a configurable auction server for human and software agents. In *Proceedings of the 2nd International Conference on Autonomous Agents*, pages 301–308. ACM, 1998.