A Formal Modeling Method for Grid Workflow Based on Concurrent Transaction Logic

Zhilin Feng, Yanming Ye, Xiaoming Liu

1 College of Zhijiang, Zhejiang University of Technology, Hangzhou 310024, China, zjuhzfzl@gmail.com
2 College of Computer Science, Zhejiang University, Hangzhou 310027, China, yeym@zju.edu.cn
3 College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430081, China, lxmspace@gmail.com
doi:10.4156/jcit.vol6. issue2.7

Abstract

Grid workflow is an important means to implement the cooperation of resources in the grid and becomes a key alternative to develop truly distributed applications for the grid platform. Grid workflow modeling is the basis of the establishment and execution of workflows. However, it is a challenging task for grid workflow modeling, due to the high degree of autonomy and heterogeneity of the cooperative organizations under distributed environments. Recently, Concurrent TRansaction logic (CTR) has become one of the most exciting methods for grid process modeling. In this paper, we present our current work for the modeling and controlling execution of grid workflow by CTR technique. We describe in detail the syntax of modeling method and summarize special communication primitives from which any process can be defined and understood by their interactive behaviors. The proposed method can provide some new functions like implementing comprehensive coordination and cooperation behaviors of different business processes to improve the stability of grid workflow system. Experimental results show that the proposed method works well for characterizing the behaviors and interactions of the workflow processes under heterogeneous environments in terms of the semantics of CTR.

Keywords: Grid Workflow, Concurrent Transaction Logic, Heterogeneous Environment

1. Introduction

In recent years, scientists and engineers are building more and more complex applications to manage and process large data sets, and execute scientific experiments on distributed resources [1, 2]. Such application scenarios require means for composing and executing complex grid computing. Grid computing has emerged as a global platform to support organizations for coordinated sharing of distributed data, applications, and processes [3]. To satisfy these demands, many efforts have been made towards the development of workflow systems for Grid computing. Grid workflow systems which are evoking a high degree of interest aim to support modeling, redesign and execution of large-scale sophisticated e-science and e-business processes [4, 5].

Although workflow applications have been extensively studied in areas such as business process modeling and web services, grid workflow model is relatively new in the Grid computing area. The workflow application scenarios often operate in dynamic and distributed environments, dealing with a large number of heterogeneous information sources with evolving contents and dynamic availability [4]. Grid workflows, in contrast to production and administrative business workflows, are normally more flexible and completely automatic. They typically rely on distributed and autonomous processes for information interaction [5]. Because the data and computing may be dispersed in a physically distributed environments(especially under heterogeneous environments), one of the key challenges for grid computing is to define a common mechanism that the grid workflow system can handle the data transfer issue and invoke the computational tools over a distributed and heterogeneous platform. This mechanism is exactly the goal that grid computing technology works to achieve in scientific
environments. Grid computing technology satisfies the requirement by providing a new computing infrastructure for large-scale resource sharing and distributed system integration.

As is known to all, current workflow products are generally intra-organizational and based on centralized architectures. Therefore, they typically lack scalability and are also not very useful for implementing heterogeneous grid computing [6]. The rise of large-scale sophisticated e-science and e-business processes highlights the fact that more and more service processes are crossing organizational boundaries [7, 8]. Thus, there needs to be a paradigm shift in technology to overcome the limitations of original workflow systems and develop grid computing systems which can work perfectly under heterogeneous environments. Heterogeneous workflow is anticipated as a supporting mechanism for grid workflow systems.

This paper aims at designing and building basic infrastructure for grid computing in the form of a heterogeneous workflow system capable of defining and enacting business processes, as well as supporting related interactive behaviors among these business processes. Such infrastructure consists of a heterogeneous workflow system acting as the underlying grid system controlling the execution of business processes. In order to implement comprehensive coordination and cooperation behaviors of different business processes, we apply Concurrent TRansaction logic (CTR) [9, 10] to define the communication primitives from which any process can be defined and understand their interactive behaviors. This paper extends sequential transaction logic with logical connectives for concurrent execution of processes. It also presents the semantic specification of grid workflow structures which are based on the extended logic. This integration of process modeling and logical functionality is reflected in the formal semantics of CTR. In particular, this approach shows its capability to describe complex, concurrent, distributed nature of heterogeneous business processes.

The rest of the paper is organized as follows. Section 2 reviews related work in the arena of grid workflow modeling. Section 3 gives the syntax and operational semantics of CTR. Section 4 contains semantic descriptions about grid workflow specification by using CTR technology. Section 5 applies this approach to a grid workflow setting. Finally, Section 6 summarizes the main results.

2. Related works

Workflow and process management has been a research topic in business management and other areas for many years. Since a grid application could be a collection of loosely coupled activities, workflow is considered as a programming model for grid applications. Related research issues include workflow specification language, workflow engine/enactor, workflow simulation, workflow scheduling, workflow monitoring, workflow editor, and so on. Grid workflow can be seen as a collection of tasks that are processed on distributed resources in a well-defined order to accomplish a specific goal. Such workflow system is essentially a set of loosely coupled service agents [4]. Typically, there are many service agents which are involved in one global workflow process under heterogeneous environments. Each of the agents has its own local workflow process. Each local workflow process is private, i.e. the corresponding service agent has full control over the local part of the workflow. These local workflow processes need to communicate because they depend on each other for the correct execution of cases. The global grid workflow consists of local workflow processes and an interaction structure. Figure 1 shows a typical example of grid workflow system which consists of five heterogeneous workflow systems.

Recently there has been an increased interest in formal modeling of grid workflow systems. Several formalized languages for the specification of grid workflows have been proposed, each of them having different origins and pursuing different goals for dealing with the unique characteristics of grid workflow. The two main formal tools used to model such workflow processes are Petri nets [11] and Process algebra [12, 13]. Many notions developed for Petri nets can be translated to process algebra and vice versa. The main differences between Petri nets and process algebra are:
- Process algebra (such as pi-calculus) are usually Turing-complete,
- Process algebra model communicating processes,
- Process algebra contains features such as name/channel passing and a limited form of mobility, which is certainly not present in Petri nets.
In some sense, workflow processes are built upon process calculus, the formal methods of computation and communication that underpin dynamic cooperative processes, as opposed to static independent processes. Pi-calculus, one branch of process calculus, has recently drawn considerable attention in the computer science community [13].

Nowadays, concurrent transaction logic (CTR) has become one of the most exciting methods for concurrent business process modeling. CTR is an extension of sequential transaction logic that was developed by Bonner and Kifer [9]. It is a deductive database language that integrates queries, updates, and transaction composition in a simple logical framework. The communication paradigm within CTR is inspired by the Pi-calculus. CTR has many of the features of process algebras. These include concurrent access to shared resources, communication between sequential processes, and the ability to isolate the inner workings of a group of processes from the outside world. Like Pi-calculus, CTR is compositional, so processes can be defined recursively in terms of sub-processes.

There are several reasons for using CTR for grid workflow modeling:

- Since the architecture of grid workflow system is highly dynamic, it is necessary to develop an efficient technology that is capable of handling dynamic behaviors across different process boundaries. It requires that the formal method should be capable of coping with those dynamic aspects, and the CTR can precisely satisfy this requirement.
- Processes in a grid workflow system are crudely concurrent. It is difficult for classical or non-classical logic to describe the concurrency among processes, but it is easy for the CTR to do so.
- A good formal method for grid workflow system should be able to express interactions among processes. CTR naturally takes the advantage of the ability to describe interactions among processes.

3. Concurrent TRansaction logic (CTR)

Central to the CTR theory is the notion of a formula. CTR formulas are intended to represent transactions that execute by querying the underlying database state and modifying that state by adding or deleting facts. Informally, executing a transaction along a sequence of database states $D_0, ..., D_n$ means that the transaction starts at state $D_0$, changes it to state $D_1$, then to $D_2$, etc., terminating in state $D_n$. Note that since the definitions of CTR formulas are stored in the workflow database, the values of these formulas can be evaluated based on the values of corresponding predicates which are stored in the workflow database.

Formally, the syntax of CTR formulas is as follows:
Definition 1 A variable is a term. If \( f \) is an n-ary function symbol and \( t_1, \ldots, t_n \) are terms, then \( f(t_1, \ldots, t_n) \) is also a term.

Definition 2 Every atomic formula, \( p(t_1, \ldots, t_n) \) is a CTR formula, where \( p \) is a predicate symbol and each \( t_i \) is a term. An atomic formula represents either an elementary update operation or a call to a complex transaction, whose behavior is defined via Horn-like rules.

Definition 3 The semantics of CTR formulas are defined over multi-paths. A multi-path is a finite sequence of workflow database states which represents the execution of a formula.

If \( M \) is a multi-path structure and \( \pi \) a multi-path, then the satisfaction of a formula \( \alpha \) is denoted by \( M, \pi \models \alpha \).

Definition 4 If \( \alpha \) is a CTR formula, then \( \neg \alpha \) is a negated formula, which represents exactly those executions that are not executions of \( \alpha \). \( M, \pi \models \neg \alpha \) if and only if \( M, \pi \not\models \alpha \).

Definition 5 If \( \alpha \) is a CTR formula, then \( \alpha \) is a negated formula, which represents executions that are not executions of \( \alpha \). \( M, \pi \models \neg \alpha \) if and only if \( M, \pi \models \alpha \), where \( \pi \) is not a multi-path. Intuitively, isolated formula means executing \( \alpha \) in isolation without interleaving with the execution of other formulas.

Definition 6 If \( \alpha \) is a CTR formula, then \( (\forall X ) \alpha \) is a quantification formula. \( M, \pi \models (\forall X ) \alpha \) if and only if \( M, \pi \models [X / t] \alpha \) for every assignment of a ground term to the variable \( X \). Here \( [X / t] \alpha \) denotes \( \alpha \) with all free occurrences of the variable \( X \) replaced by the term \( t \).

Definition 7 If \( \alpha \) and \( \beta \) are CTR formulas then \( \alpha \land \beta \) is conjunction formula, which represents executing \( \alpha \) and \( \beta \) so that the execution path will also be a valid execution of \( \beta \). \( M, \pi \models \alpha \land \beta \) if and only if \( M, \pi \models \alpha \) and \( M, \pi \models \beta \).

The conjunction formula forms the basis for representing constraints on the executions of workflows. Typically, \( \alpha \) would represent a constraint condition and \( \beta \) a workflow.

Definition 8 If \( \alpha \) and \( \beta \) are CTR formulas then \( \alpha \lor \beta \) is disjunction formula, which represents executing either \( \alpha \) or \( \beta \). \( M, \pi \models \alpha \lor \beta \) if and only if \( M, \pi \models \alpha \) or \( M, \pi \models \beta \).

Definition 9 If \( \alpha \) and \( \beta \) are CTR formulas then \( \alpha \otimes \beta \) is serial conjunction formula, which represents executing \( \alpha \) and then \( \beta \). \( M, \pi \models \alpha \otimes \beta \) if and only if \( M, \pi_1 \models \alpha \) and \( M, \pi_2 \models \beta \), for some multi-paths \( \pi_1 \), \( \pi_2 \), and \( \pi = \pi_1 \otimes \pi_2 \), where \( \otimes \) denotes concatenation of two multi-paths.

The serial conjunction formula forms the basis for representing a sequential composition of tasks in a workflow.

Definition 10 If \( \alpha \) and \( \beta \) are CTR formulas then \( \alpha \mid \beta \) is concurrent conjunction formula, which represents executing \( \alpha \) concurrently, interleaved operation of other formulas. \( M, \pi \models \alpha \mid \beta \) if and only if \( M, \pi_1 \models \alpha \) and \( M, \pi_2 \models \beta \), for some multi-paths \( \pi_1 \), \( \pi_2 \), and with an interleaving in both of the multi-path \( \pi_1 \) and \( \pi_2 \).

The serial conjunction formula forms the basis for representing a sub-process structure in a workflow schema. For example, a valid execution of the concurrent conjunction formula could be a path where one sub-process, say, \( \alpha \) starts. This execution may be interrupted by execution of \( \beta \). Execution of \( \beta \) can also be interrupted, and \( \alpha \) may be resumed. The resumed execution of \( \alpha \) may again be interrupted and \( \beta \) resumed.

In summary, the following are some examples of CTR formulas:

- \( \alpha(X) \otimes \beta(X) \): \( \alpha \) and \( \beta \) are executed serially;
- \( \alpha(X) \otimes \beta(X) : \gamma \) and \( \phi \) are executed serially, run \( \alpha \otimes \beta \) and \( \gamma \otimes \phi \) concurrently, however, \( \gamma \otimes \phi \) is atomic, so any other activity can not executed between them;
- \( \forall X[(\alpha(X) \lor (\beta(X) \lor \gamma(X)))] : \) for all \( X \), \( \alpha \) is executed successfully or \( \beta \) and \( \gamma \) execute serially.

Note that the parameter \( X \) in the above samples is used as a reference to an object in a workflow. For each parameter, it has a finite domain which represents a finite set of objects. When a parameter is instantiated to a particular value from its domain, it becomes a reference to that particular object.

Definition 11 A CTR goal is a formula of the following form:
4. Grid workflow modeling using CTR

In this section, we present some workflow specification structures which are the basis of building grid workflows from a set of tasks or existing heterogeneous sub-workflows. A grid workflow is essentially a set of loosely coupled workflow processes. Typically, there are many business partners which are involved in one grid workflow process. Each of the partners has its own local workflow process. Each local workflow process is private, i.e. the corresponding business partner has full control over the local part of the workflow. These local workflow processes need to communicate because they depend on each other for the correct execution of cases. The grid workflow process consists of local workflow processes and an interaction structure. Therefore, a grid workflow specification can be seen as a collection of computational tasks that are processed in a well-defined order to accomplish a specific goal. In a grid workflow application, tasks are related to one another via flow transition information. Formally, the syntax of grid workflow model in terms of the semantics of CTR is as follows:

**Definition 12** A grid workflow model is a tuple $W = (LP_1, LP_2, \ldots, LP_n, V_w, t_w, f_w, R_w)$, where
- $n$ is the number of local workflow processes,
- for each $k \in \{1, \ldots, n\}$, $LP_k$ is a local workflow process,
- $V_w = \{v_1, \ldots, v_{n}\}$ is a set of ports in a local process,
- $t_w(v_i) \in \{START, TASK, DECISION, SPLIT, JOIN, END\}$ indicates the type of a channel,
- $V_x$ for $X \in \{START, TASK, DECISION, SPLIT, JOIN, END\}$ denotes the subset $V_x \subseteq V_w$ of all channels of type $X$,
- $f_w : V_{\text{task}} \rightarrow A$ is the task assignment function,
- $R_w \in (V_w \times V_w)$ is a set of channels among local workflow processes.

In Definition 12, the overall processes are decomposed into many kinds of tasks that are ordered based on the dependencies among them.

**Definition 13** A task is specified by a triple $\langle P \rangle \land T \otimes \langle Q_1, \ldots, Q_r \rangle$ where $T$ is the identifier of the task, $P$ is a set of transition (precondition) for $T$, and $Q_1, \ldots, Q_r$ is a set of transition (postcondition) for $T$.

In grid workflow reference model, a task corresponds to a generic piece of work. A task is not defined for a specific business object, but for a type of objects, i.e., a task may be executed for many business objects.

**Definition 14** If task $A$ is automatic task, then its corresponding representation is:

\[
A = \text{request\_resource}[\text{resource\_id}] \cdot \text{START} \cdot \text{assigned\_resource}[\text{resource\_id}] \cdot \text{ACTION} \cdot \text{release\_resource}[\text{resource\_id}] \cdot \text{FINISH}
\]

The above definition shows that the task requests resource through the name request\_resource after task is enabled. The name resource\_id represents the identity of resource required. The name assigned\_resource is used for the receiving of assigned resource. After the resource has been assigned, the task executes operation ACTION by interacting with external application or tool. When the activity has completed, it releases resource through release\_resource. Finally, the activity informs the next activity to continue.

**Definition 15** If task $A$ is manual task, then its corresponding representation is:

\[
A = \text{request\_resource}[\text{resource\_id}] \cdot \text{START} \cdot \text{assigned\_resource}[\text{resource\_id}] \cdot \text{wait\_user}[\text{role\_id}] \cdot \text{ACTION} \cdot \text{release\_resource}[\text{resource\_id}] \cdot \text{FINISH}
\]
Definition 15 is similar to definition 14, except that manual task will wait for user’s preparation for the task through wait user and role id after resource is assigned.

**Definition 16** If task $A$ is time-triggering task, then its corresponding representation of process is:

$$
A = \text{request resource}[\text{resource id}], \text{START timer } < \text{begin time}, \text{end time}>. \text{Counter assigned resource(}
\text{resource id}), \text{ACTION release resource[resource id]}, \text{FINISH}
$$

Definition 16 is similar to definition 14, except that a time-triggering activity inquires a time counter evaluated from begin time to end time after resource is assigned. A time variable is divided into message time (absolute time when message is sent to the participant) and reply time (absolute time when reply is sent by the participant). CRT handles time variable in a more declarative way than assuming the existence of a “timer” activity.

**Definition 17** A sub-process is specified by a triple $\{P \wedge \forall Q, \ldots, Q_n\}$ where $SW$ is the identifier of the sub-process, $P$ is a set of precondition for $SW$, and $Q, \ldots, Q_n$ is a set of postcondition for $SW$.

In grid workflow reference model, a sub-process process is composed of one or more tasks following a certain order. Each task in sub-process is not isolated.

**Definition 18** A grid workflow specification consists of a set of CTR goals and CTR rules. A grid workflow schema can be made up of a collection of tasks and sub-processes. The tasks can be represented by the CTR goals to execute sequentially and concurrently, and the sub-processes can be represented by CTR rules to execute a sequence of cooperating tasks.

Intuitively, the predicate $gridworkflow(W)$ represents the routine flow of a special schema. Likewise, the predicate $subflow(W)$ represents the sub-process, and the predicate $task_i(W)$ represents the $i$th workflow task.

For example, we describe a simple grid workflow by the Definition 12–18:

$$
\begin{align*}
gridworkflow(W) & \leftarrow task_1(W) \oplus task_2(W) \oplus task_3(W) \oplus task_4(W) \\
& \quad \ominus subflow(W) \ominus task_5(W) \\
subflow(W) & \leftarrow condition(W) \wedge task_6(W) \ominus task_7(W) \\
subflow(W) & \leftarrow \neg condition(W) \wedge [task_8(W) \ominus task_9(W)]
\end{align*}
$$

The first rule indicates that, firstly the task $task_1$ should be applied to $gridworkflow(W)$, secondly $task_2$ and $task_3$ should be applied concurrently, and finally $W$ should be passed to a sub-process for further processing. The second rule indicates that the $task_5$ should be applied by using of the modality of isolation. This ensures that the $task_5$ and $task_6$ are carried out as a single transaction, i.e., as if $task_5$ were an elementary operation. The third rule shows that the sub-process $subflow(W)$ applies a series of tasks to $W$ depending on the condition of conjunction formula $condition(W)$. If $condition(W)$ is true, then $task_6$ and $task_7$ are serial applied, otherwise, $task_8$ and $task_9$ are serial applied.

In CTR based grid workflow system, the overall processes are decomposed into many kinds of tasks that are ordered based on the dependencies among them.

- **Sequential**: Tasks are executed in sequence, i.e., one task is followed by the next task.
- **Loop**: It may sometimes be necessary to execute a task or a set of tasks multiple times until a certain condition is met.
- **Parallel**: Two or more tasks are executed in parallel. Four building blocks are identified:
  1. (AND-split): enabling two or more tasks to be executed concurrently after another task has been completed.
  2. (AND-join): synchronizing the parallel flows, one task starts only after all tasks in the join have been completed.
  3. (XOR-split): activating one outgoing link from the task. Therefore, it doesn't create a situation of concurrency.
  4. (XOR-join): no synchronization is required. One task starts when any one of tasks in the join has been completed.
Definition 19 If task $\alpha$ and task $\beta$ have a sequential dependency, and the condition of routing between task $\alpha$ and task $\beta$ is always true, i.e., $\alpha \oplus [\text{Completed}(\alpha) = \text{true}] \land \beta$, where $\text{Completed}$ is a predicate condition for completing task $\alpha$, then the sequential dependency can be defined as:

$$\alpha \text{ sequential } \beta = \alpha \oplus [\text{Completed}(\alpha) = \text{true}] \land \beta = \alpha \oplus \beta$$

Definition 20 If task $\alpha$ and task $\beta$ (a set of tasks $\beta_1, ..., \beta_n$) have AND-split dependency, and the condition of routing between task $\alpha$ and task $\beta_i (i \in [1, n])$ is always true, i.e., $\alpha \oplus [\text{Completed}(\alpha) = \text{true}] \land \beta_i$, where $\text{Completed}$ is a predicate condition for completing task $\alpha$, then the AND-split dependency can be defined as:

$$\alpha \text{ AND - split } \beta = \alpha \oplus \{[[\text{Completed}(\alpha) = \text{true}] \land \beta_i], ..., [[\text{Completed}(\alpha) = \text{true}] \land \beta_n]\} = \alpha \oplus \{\beta_i, ..., \beta_n\}$$

Definition 21 If task $\alpha$ (a set of tasks $\alpha_1, ..., \alpha_n$) and task $\beta$ have AND-join dependency, and the condition of routing between task $\alpha_i (i \in [1, n])$ and task $\beta$ is always true, i.e., $\alpha \land [\text{Completed}(\beta) = \text{true}] \land \alpha$, where $\text{Completed}$ is a predicate condition for completing task $\alpha$, then the AND-join dependency can be defined as:

$$\alpha \text{ AND - join } \beta = \{[\alpha_i \land \text{Completed}(\beta) = \text{true}], ..., [\alpha_n \land \text{Completed}(\beta) = \text{true}]\} \land \beta = \{\alpha_i, ..., \alpha_n\} \land \beta$$

Definition 22 If task $\alpha$ and task $\beta$ (a set of tasks $\beta_1, ..., \beta_n$) have XOR-split dependency, and the condition of routing between task $\alpha$ and task $\beta_i (i \in [1, n])$ is not always true, i.e., $\text{Completed}(\alpha) = \text{true}$ is only for a certain task $\beta_i$, and not for the others: $\exists j \alpha \oplus [\text{Completed}(\alpha) = \text{true}] \land \beta_j$ and $\alpha \oplus [\text{Completed}(\alpha) = \text{false}] \land \beta_i (i \neq j)$, then the XOR-split dependency can be defined as:

$$\alpha \text{ XOR - split } \beta = \exists j \alpha \oplus \{[[\text{Completed}(\alpha) = \text{true}] \land \beta_i], ..., [[\text{Completed}(\alpha) = \text{false}] \land \beta_n]\} \lor \beta = \exists j \alpha \oplus \{\beta_i, ..., \beta_n\}$$

Definition 23 If task $\alpha$ (a set of tasks $\alpha_1, ..., \alpha_n$) and task $\beta$ have XOR-join dependency, and the condition of routing between task $\alpha_i (i \in [1, n])$ and task $\alpha$ is not always true, i.e., $\text{Completed}(\alpha) = \text{true}$ is only for a certain task $\alpha_j$, and not for the others: $\exists j \alpha_i \land [\text{Completed}(\beta) = \text{true}] \land \alpha_j$ and $\alpha_j \land [\text{Completed}(\beta) = \text{false}] \land \alpha_i (i \neq j)$, where $\text{Completed}$ is a predicate condition for completing task $\beta$, then the XOR-join dependency can be defined as:

$$\alpha \text{ XOR - join } \beta = \exists j \{[\alpha_i \land [\text{Completed}(\beta) = \text{true}] \lor \alpha_j \land [\text{Completed}(\beta) = \text{true}]\} \lor \beta = \exists j \alpha_i \lor \alpha_j \lor \beta$$

Definition 24 If the decision expression $\psi$ is true, then task $\alpha$ will be executed; otherwise task $\beta$ will be started. Boolean expressions for completing task $\alpha$ or task $\beta$ can be defined as:

$$\text{def } \psi \text{ then } \alpha \text{ else } \beta = (v, a, b)[\text{Decision}_{\text{exp}}(\psi), \text{START}((\psi = \text{TRUE})a + (\psi = \text{FALSE})b)]((a / \text{start})\alpha + (b / \text{start})\beta)$$

In the above definition, the channel $\text{decision}_{\text{exp}}$ provides a Boolean condition expression $\psi$ to be evaluated. The private port $a$ and $b$ instruct which task ($\alpha$ or $\beta$) should be selected to execute.

5. Experiments

In this section, we introduce a real grid workflow system implemented by CRT mechanism. It is a loosely coupled workflow system, which consists of four heterogeneous workflows: Customer, Producer, Supplier_A and Supplier_B. For a process type called "loosely coupled", which allows a process to be split up into parts and be executed in parallel. The Customer orders a product by sending an order for a product to the Producer. To produce the ordered product, the Producer orders the products needed for production (type $A$ and type $B$). Then the Customer is informed that the order has been accepted. The Supplier_A produces products of type $A$, the Supplier_B produces products of type...
B. After both products have been delivered, they are assembled into a product which is delivered to the Customer. After delivery an invoice is sent which is then paid by the Customer.

Figure 2 shows the modeling procedure of above grid workflow system. Figure 3 shows how communication procedures among grid workflow processes are handled. In Figure 3, each of four heterogeneous processes is enclosed with an ellipse line respectively. Furthermore, the communication links among processes are labeled with bold lines to see more clearly.

In Figure 3, it can be easily seen that the process $P_{customer}$ shares five external channels ($order_A$, Notification, Delivery, Invoice, and Payment) with the process $P_{producer}$, and the process $P_{producer}$ shares four channels ($Order\_SupplierA$, $Order\_SupplierB$, Delivery_A, and Delivery_B) with the process $P_{supplier\_A}$ and $P_{supplier\_B}$.

Figure 2. The modeling procedure of a grid workflow system

Figure 3. Communication procedures under distributed environments
In $P_{customer}$, the task $T_1$, $T_3$, and $T_5$ are automatic tasks, task subflow is a subflow task, and the others are manual tasks. By the Definition 18~23, the $P_{customer}$ can be represented as the CTR goal as follows:

$$P_{customer} \leftarrow [\text{Send Order} \land \text{Producer} \land T(P_{customer})] \otimes \text{[order A \land subflow}(P_{customer})] \otimes \text{[Receive Notification} \land T(P_{customer})] \otimes \text{[Receive Place} \land T(P_{customer})] \otimes \text{Send Invoice} \land T(P_{customer})]$$

$$\land T_1(P_{customer}) \otimes \text{[Receive Notification} \land T_3(P_{customer})] \otimes \text{[Receive Delivery} \land \text{Producer} \land T_1(P_{customer})] \otimes \text{Send _Invoice} \land \text{T_3(Pcustomer)}$$

$$\text{subflow(Pcustomer)} \leftarrow [\text{Receive _Order} \land \text{Producer} \land T(P_{customer})] \otimes \text{[Send _Order} \land \text{SupplierA} \land T(P_{customer})] \otimes \text{[Notify} \land \text{T_3(Pcustomer)}]$$

Formalization descriptions of the three other processes are given as follows:

$$\text{P_{producer}} = [\text{Receive _Order} \land \text{Producer} \land T(P_{producer})] \otimes \text{[Send _Order} \land \text{SupplierA} \land T(P_{producer})]$$

$$\text{\lor [Send _Order} \land \text{SupplierB} \land T(P_{producer})] \otimes \text{[Notify} \land \text{T_5(Pcustomer)}]$$

$$\text{\lor [Receive _Delivery} \land \text{SupplierA} \land T_1(P_{producer})] \lor \text{[Receive _Delivery} \land \text{Producer} \land T_2(P_{producer})]$$

$$\otimes \text{[Send _Invoice} \land \text{T_3(Pcustomer)}]$$

$$\text{subflow(Pproducer)} \leftarrow [\text{Receive _Notification} \land T(P_{customer})] \land$$

$$\text{subflow(Pproducer)} \leftarrow [\text{Receive _Invoice} \land T(P_{customer})] \otimes \text{[Pay} \land \text{T_3(Pcustomer)}]$$

Thus, the overall grid workflow system is the parallel composition of four processes:

$$\text{grid _workflow(W)} \leftarrow P_{customer} \mid P_{producer} \mid P_{supplier A} \mid P_{supplier B}$$

Next, we use CRT to describe the behavior of tasks in Figure 3. For example, we give the following descriptions of activities in $P_{producer}$ according to definition 14 to 16. Due to space limitation, the specification of activities in three other processes is omitted.

$$T_1(\text{Send _Order} \land \text{SupplierA}) \leftarrow \text{request resource}[\text{resource id}, \text{START}] \text{assigned resource}$$

$$(\text{resource id}), \text{Send _Order} \land \text{SupplierA}, \text{Action}, \text{release resource}[\text{resource id}, \text{FINISH}]$$

$$T_2(\text{Notify}) \leftarrow \text{request resource}[\text{resource id}, \text{START}] \text{assigned resource}[\text{resource id}]$$

$$\text{wait user[role id], Notify, Action}, \text{release resource}[\text{resource id}, \text{FINISH}]$$

$$T_3(\text{Send _Delivery}) \leftarrow \text{request resource}[\text{resource id}, \text{START}] \text{timer < begin time, end time >}$$

$$\text{Counter, assigned resource}[\text{resource id}], \text{Send _Delivery}, \text{Action}, \text{release resource}[\text{resource id}, \text{FINISH}]$$

By the Definition 20~23, the task dependencies of above four processes can be represented as the CTR formulas as follows:

$$T_1 \text{ XOR - split } (T_1, T_3) = [\text{Send _Order} \land \text{SupplierA} \land T(P_{producer})] \otimes \text{T_3(Pcustomer)}$$

$$\lor [\text{Send _Order} \land \text{SupplierB} \land T(P_{producer})] \otimes \text{T_5(Pcustomer)}]$$

$$T_2 \text{ XOR - join } T_2 = [\text{Notify} \land T(P_{producer})] \otimes \text{T_5(Pcustomer)}] \lor [\text{Notify} \land T_5(P_{producer})] \otimes \text{T_3(Pcustomer)}]$$

$$T_3 \text{ XOR - split (T_3, T_5)} = [\text{Receive _Delivery} \land \text{SupplierA} \land T(P_{producer})] \otimes \text{T_3(Pcustomer)}]$$

$$\lor [\text{Receive _Delivery} \land \text{SupplierB} \land T(P_{producer})] \otimes \text{T_5(Pcustomer)}]$$

$$T_1 \text{ XOR - join } T_1 = [\text{Send _Delivery} \land T(P_{producer})] \otimes \text{T_5(Pcustomer)}]$$
6. Conclusions

In this paper, we present a new formalizing method for modeling and controlling the execution of grid workflow system by CTR technique. Using the proposed method, we design a grid workflow specification language based on the CTR technology and develop a prototype of process design system. This approach is well suited for sporadic communication in large-scale sophisticated e-science and e-business processes. Future work includes some issues of improvement of grid workflow mechanism on efficiency, security and fault tolerance.

7. Acknowledgement

The work has been supported by the Important National Science & Technology Specific Project, China(2009ZX01043-003-003), the National Natural Science Foundation of China under grant(No. 60703042, 61070343), the Research and Application Plan of Commonweal Technology in Zhejiang Province (No. 2010C31027).

8. References


