# TRANSACTION-ORIENTED ENGINEERING DESIGN AND SPECIFICATION: A MULTI-AGENT APPROACH

**Muhammad Younas**[1]
**Kuo-Ming Chao**[1]
**Rachid Anane**[1]
**Anne James**[1]
**Chen-Fang Tsai**[2]
[1]Distributed Systems and Modelling Research Group
School of Mathematical and Information Sciences, Coventry University, Coventry, CV1 5FB, UK
{m.younas, k.chao, r.anane, a.james}@coventry.ac.uk
[2]Department of Industrial Management, Aletheia University, ?, Taiwan
Tsai@email.au.edu.tw

*Engineering design activities require fault tolerance and concurrent access to shared resources such as databases and Web servers. Such activities are generally dynamic, cooperative, long-lived, interactive and non-prescriptive. We propose a new multi-agent transaction model, which is based on extended transactions and multi-agent technologies. The novelty of this model is that it automatically customises transactions to the requirements of design activities. In addition, this model is believed to improve concurrency and fault tolerance, facilitate interaction between and co-operation among the participating systems involved in design activities. The proposed model is formally specified using CCS (Calculus of Communicating Systems) language. Formalism is crucial to model the correctness, reliability, and recovery of multi-agents transactions, given the complex and unreliable nature of the Web-based design activities.*

Keywords: Transactions, Multi-agents, Design activities, Specification, CCS.

## 1. Introduction

Complex engineering design (e.g., offshore oil platforms, aeroplanes, ships) requires collaborative work among diverse design disciplines. Such design environments often involve a number of different organisations that specialise in different design disciplines, which are distributed across a network. The Web provides an infrastructure for design activities to be geographically distributed among different teams and design systems. In such highly dynamic environment the nature of the work and the requirements may change continuously. More importantly, these activities require consistent and concurrent access to shared resources (e.g., databases, Web servers, etc.), and also require fault tolerance. This motivates the need for providing suitable techniques to facilitate this kind of cooperative activities.

Existing design approaches apply Extended Transaction Models (ETM) to design activities [8, 15, 6] (Kaiser, 1994, Nagi, 1999, Yang, *et al*., 1999). ETM relax isolation and atomicity properties of the ACID (Atomic, Consistent, Isolated, Durable) transactions [5], and can provide design activities with concurrency and consistency aspects. However, one of the major problems with current approaches is that users have to specify in advance a particular ETM for the design activity.

Similarly, multi-agents are also used in design activities [13, 12]. The role of agents in design activities is to support communication and to improve co-ordination among design systems. However, transaction-like facilities (e.g., ensuring concurrent and consistent access to shared resources [9]) are not well addressed in multi-agent systems.

Our research identifies that the exclusive use of one of the above technologies (i.e., transactions or multi-agent) remains ineffective in design activities. In this paper we present a new model that addresses the issues of distributed design activities by combining multi-agent technology with ETM. One distinguishing feature of the proposed model is that it provides a facility for automatically customising a variety of ETM to the requirements of design activities. Existing approaches [6, 15, 1] require users to specify the transaction models that suit their needs. The proposed model is also believed to improve concurrency and fault tolerance, facilitate interaction between and co-operation among participating systems.

The proposed model is formally specified using the CCS (Calculus of Communicating Systems) specification language. Formalisation is crucial to the modelling of the correctness, reliability, and recovery of multi-agents transactions, given the complex, unreliable, and non-deterministic nature of distributed design activities. The formal specification of the multi-agent transaction system is also important, because of the complexity of the environment and the nature of the interactions among various design systems. One interesting feature of the formal specification is that it guarantees the correct functionality of a system by establishing correctness criteria against which the implementation of the system can be verified and validated [18, 17].

The remainder of the paper is structured as follows. Section 2 identifies the requirements of design activities. Section 3 reviews current approaches to design activities and establishes their limitations. Section 4 presents the proposed model. Section 5 formally specifies the proposed model using CCS formalism. Finally, Section 6 concludes the paper and points to further work.

## 2. Requirements

The requirements refer to the design of large and complex engineering products (e.g., offshore oil platforms, aeroplanes, ships). Such environments often involve a number of different organisations that specialise in different design disciplines. This requires multiple design systems to co-operate. The participating systems could be semi automated or fully automated design systems with data store facilities. They should be able to operate concurrently in order to optimise the use of required design resources. Such design activities are characterised by long duration, co-operation and negotiation, non-prescriptive developments and recovery from failures. We discuss these characteristics within the context of the requirements of distributed design activities so as to define the scope for this research.

### 2.1 Long Duration

Transactions in a design environment are generally of long duration [6]. These transactions may run for hours or days to process design activities. In particular a design activity may require deliberation from humans. It is difficult to determine in advance the execution time of such transactions. Thus it is required to ensure that a particular data object is not blocked unnecessarily until a transaction is committed. For example, in designing an offshore oil platform, the cost engineer issues a design change that requires a semi-automated process flow design system and an automated layout design system to respond to the change [3]. The layout design system may take a few minutes to propose a new layout, but the process flow designer may take days to reflect the change. The layout system cannot commit the transaction until the process flow designer has completed the change. Thus it is required to commit the completed part of the transaction so as to ensure the early release of system resources.

### 2.2. Co-operation

Co-operation among design systems is a prerequisite for the accomplishment of various design tasks [7, 6]. Such systems generally require the sharing of different data objects of the design activities. Thus mechanisms are required to facilitate the exchange of intermediate results, while at the same time guaranteeing that the consistency of these results is maintained during the exchanges. For example, the safety and layout systems might be working on a location and orientation of a hazardous equipment item such as a compressor. This requires both parties to modify two parts of the same data object concurrently, with the intent of integrating these parts to create a new version of the design. In this situation, the systems might need to look at each other's work to ensure that they are not modifying two parts in a way that would lead them to have conflicts. Moreover, for some design activities, it is necessary that the participating systems develop a transient cooperation that may exist for a particular task, or for a limited time. Such transient cooperation needs to be formed dynamically with different participating parties in order to meet different requirements.

### 2.3. Non-prescription

Design activities are generally interactive and non-prescriptive. Non-prescription implies that the system cannot determine in advance the nature of the transactions involved in design activities. That is, the system may not pre-determine that transactions may cooperate, or that transactions may violate the consistency of data objects, except by actually executing them. For instance, a generic connector can be used to connect various families of valves, pipes, and other equipments. The connector can be designed in the first place without considering specific applications. However, it can be used whenever it is required. Thus to support non-prescriptive design tasks, the system should be able to start a transaction, interactively execute operations within it, dynamically restructure it, if required, and then commit or abort it at any time.

### 2.4. Failure Recovery

Failures in distributed systems such as the Web may be caused by various factors that include Internet communication failures or systems failures of Web servers for instance. These failures severely affect design activities by interrupting their progress. Such interruption may result in frequent roll back. However, frequent roll back of long-lived design transactions is not acceptable as a valuable piece of completed work might be lost. Instead, a transaction should be able to proceed (and eventually succeed) even if it partially completes. For example, the process flow engineer may propose a high cost equipment (e.g., condensate vessel), which may not meet the cost engineer's requirements. Thus the roll back should only occur for the part of the transaction in the process flow design, and not in the layout solution, as it may still be valid.

## 3. Related Work

Classical database transactions strictly follow the ACID (Atomic, Consistent, Isolated, Durable) correctness criteria [5]. Although ACID criteria are useful in maintaining data consistency, they are inappropriate for some applications. For example, these criteria do not suit the classical federated databases, which are characterised by their autonomy and heterogeneity [23, 24]. That is, to enforce ACID criteria in federated databases, their autonomy and heterogeneity may be compromised. Similarly, the isolation and atomic policy (of ACID criteria) does not suit the characteristics of design activities — which are cooperative, dynamic, long-lived, interactive, and non-prescriptive. To overcome the limitations of ACID transactions, numerous Extended Transaction Models (ETM) have been proposed [5, 8, 15, 6, 10] for design activities. ETM generally focus on the relaxation of isolation and atomicity properties. In [15] an ETM-based system, called Jpernlite, is developed for Web-enabled software engineering applications. Jpernlite mainly concerns the concurrency control mechanism of designed activities. However, it does not fully support the characteristics of design activities. In particular, it fails to provide support for the dynamic restructuring of transactions. This restructuring is usually achieved through the Split and Join transaction models [6, 10]. The Split transaction model splits an active transaction, $T_i$, into multiple transactions, $T_i$ and $T_j$, by delegating some of the actions of $T_i$ to $T_j$ ($T_i$ is assumed to preserve its identity after the split). $T_i$ and $T_j$ can commit or abort independently. The Join transaction model merges multiple independent transactions, $T_i$ and $T_j$, into a single transaction, $T_k$.

Multi-agent systems (MAS) are also used in design activities [13, 3]. An agent is an intelligent and autonomous system that can control its own behaviour and respond to the requests from other agents in order to achieve a common goal [12]. An agent incorporates a reasoning mechanism such as Belief Desire and Intention (BDI) model — which is a procedural reasoning method that enables agents to deal with highly dynamic activities such as design. The BDI mental model has become increasingly popular and has been used in many applications. BDI provides a clear conceptual model that integrates seamlessly reactive and deliberative behaviours. The mental categories form the basis for a mental state: beliefs, desires and intentions. The beliefs represent the information that the agent believes is currently true. This information may concern the agent itself, other agents or the environment. The desires or goals are the states that the agent wishes to achieve. An intention involves the selection of a particular plan and a commitment to its execution. An agent can be aware of its environment and can refer to its own goal to select appropriate plans and carry out related actions in order to reach a desired state. Thus, the BDI model provides agents with a mechanism for representing knowledge (beliefs) and for modelling its behaviour (semantics). A dynamic environment often incorporates unexpected situations. It is therefore necessary for agents to handle exceptions so as to ensure that the system is consistent and thus avoid abnormal termination.

The role of agents in design activities is to support communication and to improve co-ordination among design systems. The aim is to present an integrated environment for the exchange of information and knowledge between participating design systems. However, transaction-like facilities (e.g., ensuring concurrent and consistent access to shared resources [9]) are not well addressed in multiple design agent systems.

As stated earlier, the exclusive use of the above technologies (i.e., transaction or multi-agent systems) remains ineffective in design activities. One of the major problems with current ETM systems [15, 10] is the decision-making facility. For instance, it is difficult to determine where to split and join transactions, or where to compensate or replace transactions. Furthermore, some activities may trigger other activities that may require cooperation from different heterogeneous design disciplines (e.g., in offshore oil platform). These disciplines may need to cooperate and negotiate before performing the requested tasks that pertain to design activities. For example, a pump needs to be upgraded in order to produce bigger pressure than the current one. This implies that the size of the pump will increase, so the layout system needs to be partially re-designed. However, the proposed pump may require excessive usage of electricity, which the current power generator cannot supply. Thus, the change of the power generator is inevitable, if the new pump is to be installed. In such environment, activities are non-prescriptive and thus it is difficult to model them in advance. This requires context understanding and well-defined semantics. Agents have the capability to support decision-making, provide cooperation and negotiation between different heterogeneous disciplines, and model non-prescriptive activities.

The above discussion makes it imperative, therefore, to combine multi-agent and transaction technologies so as to provide newer services in different application domains. The combination of multi-agent systems with transaction technologies has been investigated in [2, 4, 8, 11]. However, [2, 4, 8] mainly concern e-commerce applications. For example, [4] uses multi-agents to model dynamic Web transactions in the e-commerce applications, and introduces a scripting language, called Multi-Agent Processing Language (MAPL), for specifying the behaviour of agents and their interaction. Similarly, in [2] agent technology is applied to model cooperative transactions in e-commerce applications, whereas [8] applies transaction techniques to multi-agent systems in scheduling production orders in a manufacturing environment. It applies open-nested transactions to schedule different requests made for a particular product. This approach also corresponds to e-commerce applications. One notable contribution was presented in [11] where some requirements for multi-agent transactions in design activities were laid down. However, this approach is limited to the definition of requirements as no solution is proposed.

## 4. Multi-Agent Transaction Model

This section first presents an overview of the proposed approach through a case study of a design application. This is followed by the description of the dependencies that exist between the transactions of a design activity.

### 4.1 Overview of the Proposed Model

The proposed model is a combination of multi-agents [13, 12] with Extended Transaction Models (ETM) such as split, join, and open-nested transactions [14, 6, 10]. The model is diagrammatically shown in Figure 1, in which multiple agents are used to execute transactions in design activities. An agent is called a *base agent* (denoted $\text{Agent}_B$) if it starts the execution of the root (or main) transaction, T. Other agents are called participating agents, denoted as $\text{Agent}_P$. A particular participating agent is represented as $\text{Agent}_{Pi}$ (i = 1 … n). In Figure 1, $\text{Agent}_B$ is a base agent as it starts executing transaction T, which performs the tasks of a particular design activity.

The model is further illustrated through the following example. Let us consider the design activity of an offshore oil platform that performs a pump replacement so as to produce more pressure. The task of the design activity is to replace the old pump with two new pumps in order to produce the desired pressure. The activity, named A:ReplacePump, is diagrammatically shown in Figure 2, and is comprised of other subactivities that include: B:NewPump1, C:NewPump2, D:AllocatePipesPump1, E:ConnectorPump1, F:AllocatePipesPump2, G:ConnectorPump2. Activity D in turn has a number of subtasks, such as getting new pipes (D1:NewPipes) or relocating existing pipes (D2:ExistingPipes). Similarly activity F in turn has a number of subtasks, such as getting new pipes (F1:NewPipes) or relocating existing pipes (F2:ExistingPipes). Activity J:Installation installs the new pumps. Activity K:Testing tests the new setup and may require changes in the new installation if it does not work properly. In the proposed model these sub-activities are differentiated between vital and non-vital sub-activities. Non-vital sub-activities are defined as the activities whose failure does not result in the failure of the overall design activity such as A:ReplacePump. Non-vital sub-activities are also replaceable. Vital sub-activities are those whose failure results in the failure of the overall design activity, such as A:ReplacePump.

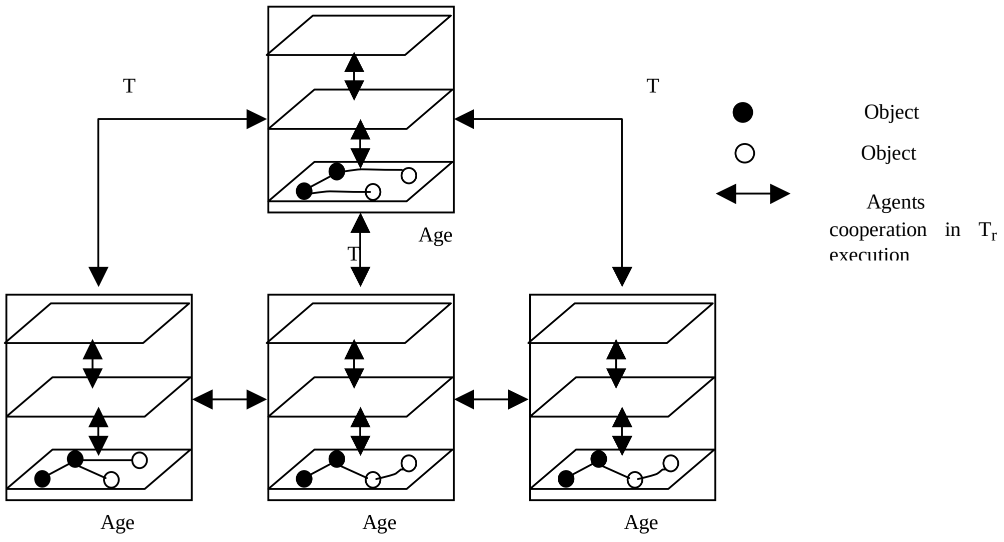

Figure 1. A Generalised Architecture for Multi-Agent

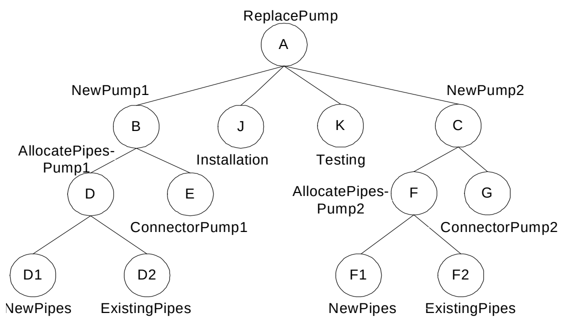

Figure 2. Example Design Activity

In order to successfully perform the pump replacement the system must complete all the vital sub-activities. However, if any of the vital sub-activities cannot complete, then the remaining (completed or uncompleted vital and non-vital) sub-activities must be cancelled, as partial arrangements are not be acceptable to the design engineer. Such consistent execution of design activity can be achieved if agents process each sub-activity according to transactional correctness criteria. Each sub-activity (e.g., AllocatePipesPump1) can therefore be modelled as a component transaction of the design activity. We use transaction, $t_i$ (i = 1.. n), to represent the above design activities such that $t_1$ = B, $t_2$ = C, $t_3$ = D, $t_4$ = E, $t_5$ = $D_1$, $t_6$ = $D_2$, $t_7$ = F, $t_8$ = G, $t_9$ = $F_1$, $t_{10}$ = $F_2$, $t_{11}$ = J, $t_{12}$ = K. Thus the overall design activity A:ReplacePump is represented by transaction Ta, such that,

$T_a = \{ t_1, t_2, t_3, ..., t_{12}\}$.

Since transaction $T_a$ is composed of different types of component transactions, it may require different ETMs to execute the component transactions. That is, some component transactions may follow the correctness criteria of the open-nested transaction model, while other may use the split transaction model. For example, the component transactions $t_1$ (for NewPump1) and $t_2$ (for NewPump2) can be executed using the split transaction model by splitting them into two independent transactions. They can commit or abort irrespective of each other. Similarly, the component transactions $t_7$, $t_8$, $t_9$, and $t_{10}$ are executed using alternative transaction model. That is, if the existing pipes cannot be relocated by executing transactions $t_8$ and $t_{10}$, then new pipes are required, thus leading to the execution of the transactions $t_7$ and $t_9$. These transactions are allowed to unilaterally commit by following open-nested transaction model that enforces semantic atomicity by relaxing the traditional atomicity and isolation properties. Further, transactions $t_4$ and $t_6$ will be required to use join transaction model such that it connects the pipes required by the two new pumps.

Different agents need to cooperate in the execution of the transaction $T_a$ so as to achieve the goal of pump replacement. Agents customise ETMs using their BDI model, which enables them to select appropriate plans (or partial plans) in order to achieve a desired goal [12]. To execute such transactions agents incorporate transaction management primitives such as commit, abort, split, join, compensate, and replace. Agents keep track of all the transactions by storing the necessary information in their beliefs. They make use of the plans (or partial plans) associated with intentions in order to execute the transactions.

Furthermore, an awareness by agents of the dependencies between transactions is required in order to enable them to coordinate their activities and to select an appropriate ETM for the execution of transactions. The following subsection describes some of the possible dependencies.

### 4.2 Dependencies

Dependencies define different constraints on the execution of transactions, and facilitate the process of specifying and reasoning about the behaviour of transactions. Dependencies are generally expressed in terms of primitive operations such as commit, abort, and split [16].

Let $T_i$ and $T_j$ be two transactions and $P_i$, $P_j$ be the corresponding plans of agent's intention, I. Let agents execute transaction $T_i$ according to plan $P_i$ and transaction $T_j$ according to Plan $P_j$. Depending on the nature of the design activity various types of dependencies may exist between $P_i$ and $P_j$. Following are some of dependencies that may occur between plans $P_i$ and $P_j$. These dependencies correspond to those defined in [16].

- **Commit:** The commit dependency, denoted as **(**$P_j$ ***Com-D*** $P_i$**),** states that $P_j$ can only be committed if $P_i$ is committed. In other words, the commit of $P_i$ precedes the commit of $P_j$.

$$(\mathrm{commit}(P_i) \in I, (\mathrm{commit}(P_j) \in I) \Rightarrow (\mathrm{commit}(P_i) \rightarrow (\mathrm{commit}(P_j))$$

For example, the plan (for ConnectorPumps) can only be committed if the plan (for AllocatePipesPump) is committed.

- **Abort:** This dependency, denoted as **(**$P_i$ ***Abt-D*** $P_j$**)** states that if $P_i$ aborts (fails to complete) then $P_j$ must be aborted.

$$(\mathrm{abort}(P_i) \in I) \Rightarrow (\mathrm{abort}(P_j) \in I)$$

For exmaple, if the plan (for NewPump1) is aborted then the plan (for NewPump2) must be aborted as a single pump will not do the desired job.

- **Begin-Abort:** This is denoted as **(**$P_i$ ***B-Abt-D*** $P_j$**).** It requires that $P_i$ cannot begin execution until $P_j$ aborts.

$$(begin(P_i) \in I) \Rightarrow (abort(P_j) \rightarrow begin(P_i))$$

This dependency describes the relationship between the alternative plans. For example, if the plan (for NewPipes) fails to complete then its alternative plan (for ExistingPipes) will be started to allocat the existing pipes.

- **Data**: This is denoted as **(**$P_i$ ***Dat-D*** $P_j$**).** $P_i$ is data dependent on $P_j$ if they share the same data object (e.g., using same connector). Thus depending on the situation either $P_i$ precedes $P_j$ or $P_j$ precedes $P_i$ in accessing the shared data object, *ob*.

$$(P_i\ (ob) \in I) \wedge P_j\ (ob) \in I) \Rightarrow (P_i\ (ob) \rightarrow P_i(ob)) \vee ((P_j\ (ob) \rightarrow P_i(ob))$$

## 5. Formal Specification of Multi-Agent Transactions

The importance and significance of formal specifications is underlined by their applications to various agent-based systems. These include, for example, [19] and [20] that use Z specification language, and [26] that use Petri nets to formally specify agent-based systems.

In this section a formal specification of the proposed model is given using CCS. CCS is chosen due to its simplicity and the wide support for the distributed and communicating systems such as multi-agents systems [25]. A brief introduction to CCS is presented before proceeding with the formalisation. The description of the relative merits of the formal specification language is, however, beyond the scope of this paper.

### 5.1 Calculus of Communicating System (CCS)

CCS is an algebra for specifying and reasoning about concurrent, distributed, and communicating systems [21, 22]. CCS provides a set of operators to control the occurrence of actions. For example, an agent can be defined to perform a set of actions {decision, send-message}, i.e., to make a decision and then send a message to another agent.

The most widely used CCS notations are summarised in the following Table 1. See [21, 22] for more information on CCS.

**Table 1. Description of the CCS Notations**

| **Notations** | **Description** |
|---|---|
| **.** | Prefix operator: For example, the process a **.** Q can perform action a and then behaves like Q. |
| $\vert$ | Composition operator: It represents the parallel composition of agents, P and Q, represented as P$\vert$Q |
| $\overline{\text{action}}$ | Output action. For example, the actions receive (input). $\overline{\text{send}}$ (output) receives input |

|  |  |
|---|---|
|  | and then sends the output . |
| $\underline{\text{def}}$ | This defines the behaviour of an agent. For example, Agent $\overset{\text{def}}{=}$ receive (input). $\overline{\text{send}}$ (output) |
| [] | Re-labelling operator: It is used to define several replicas of an agent. For example, the following process P $\overset{\text{def}}{=}$ Q [decisionp / $\text{decision}_q$] defines an agent P, which is identical to Q but uses the name $\text{decision}_p$ wherever Q uses the name $\text{decision}_q$. Relabelling is very useful when several replicas of an agent are used in defining a complex behaviour. |
| + | Summation operator: It describes a (non deterministic) choice. For example, the process commit(T) + abort(T) will either take commit action or abort action for transaction, T. |
| $\sum$ Ei | Summation: It represents the sum of all expressions $E_i$. |

## 5.2 Formal Specification

The formalisation of the systems is given in terms of agents and the components of the BDI model. For this purpose we assume that different agents (denoted as $\text{Agent}_1$, $\text{Agent}_2$, … $\text{Agent}_n$) are involved in processing $T_a$ (Pump replacement). These agents must cooperate (synchronise) with each other in order to process $T_a$. Using CCS notations the following multi-agent transaction system (denoted MTS) is defined for the design activities such as $T_a$.

$$\text{MTS} \overset{\text{def}}{=} (\text{Agent}_1 \mid \text{Agent}_2 \mid \dots \mid \text{Agent}_n)$$

The CCS composition operator | defines the interaction between agents so that they can collaborate in executing $T_a$. As interaction in CCS is synchronous, all the agents must be willing at the same time to interact. Below we provide the formal specifications of the base agents (e.g., $\text{Agent}_B$), and the participating agents (e.g., $\text{Agent}_p$) involved in the execution of the transactions.

### 5.2.1 Base Agents

Initially, $\text{Agent}_B$ (bases agent) receives a request from MTS to perform a task (e.g., replacing a pump). $\text{Agent}_B$ executes a transaction, say $T_a$, to accomplish the task. $\text{Agent}_B$ first updates its beliefs database with information pertaining to transaction $T_a$ and its component transactions $t_i$. This is formally modelled as follows:

$$\text{Agent}'_B (T_a) \overset{\text{def}}{=} \overline{\text{update}} \, (\text{Belief}, t_i, T_a) \, . \, \text{Plan} \, (n, T_a)$$

After updating the belief, $\text{Agent}_B$ selects the plans according to Plan(n, $T_a$), as follow.

$$\text{Plan} \, (n , T_a) \overset{\text{def}}{=} \textbf{if } n = \text{no-of-CT } \textbf{then } \text{Check-depend}(n, I)$$
$$\textbf{else} \sum_{i=1}^{\text{no-of-}CT} pt_i = \text{plan}(p_i, t_i) \, . \, \text{Plan} \, (n + 1, T_a)$$

Agent$_B$ constructs different plans, $pt_i$, with respect to the component transactions $t_i \in T_a$ (where no-of-CT represents the number of component transactions). In general, each plan, $pt_i$, corresponds to one component transaction, $t_i$. However, a plan, $pt_i$, can be associated with many component transactions. The set of all possible plans, $pt_i$, of intention I is represented as *SP*. After constructing the plans, Agent$_B$ takes the action Check-depend(n, I) to check the dependencies between the plans of the intention I (where 'n' is the number of possible plans of intention I).

$$\begin{aligned}\text{Check-depend}(n, I) \overset{\text{def}}{=} &\ \textbf{If } n = \text{no-of-plans } \textbf{then } \text{Execute}(n, I)\\ &\ \textbf{else } ((\text{depend}(SP^{/}, SP, I)\ \mathbf{.}\ \overline{\text{update}}(\text{Belief}, \text{dType})) +\\ &\ ((\neg\ \text{depend}(SP^{//}, SP, I)\ \mathbf{.}\ \overline{\text{update}}(\text{Belief}))\ \mathbf{.}\ \text{Check-depend}(n + 1, I)\end{aligned}$$

where $SP^{/} \subseteq SP$ is subset of plans which are dependent and $SP^{//} \subseteq SP$ is the subset of plans which are non-dependent.

If there is a dependency between a set of plans *SP*, the agent updates its belief according to the type of dependency, dType (see 4.2 for different types of dependencies). However, if there is no dependency then the agent updates its belief that the plans are independent. After determining the dependencies the agent executes the plans according to Execute(n, I).

$$\begin{aligned}\text{Execute}(n, I) \overset{\text{def}}{=} &\ \textbf{if } n = \text{no-of-plans } \textbf{then } \text{Terminate } (pt_i, I)\\ \textbf{else}\ &\ (\text{Split } (SP^{/}, SP, I)) + (\text{Join } (SP^{//}, SP, I)) + \overline{\text{ex-process}}(pt_i) + \text{loc-process}(pt_i)\ \mathbf{.}\\ &\ \text{Execute}(n + 1, I)\end{aligned}$$

Agent$_B$ may split plans into independent plans if they have no dependency. The advantage of the split is to increase concurrency by ensuring the early commit of the completed plans of a design activity. For example, plans $pt_1$ (activity B) and $pt_2$ (activity B) can split. Thus they can execute and commit independently. Similarly, Agent$_B$ may join plans if they have some dependencies. The advantage of join is that certain plans may be required to merge so as to perform a desired task. For example, plan $pt_4$ (activity E) is used to connect different pipes. Each plan, $pt_i$, is either externally processed at some other Agent$_p$ by following $\overline{\text{ex-process}}(pt_i)$, or locally processed at Agent$_B$ by following loc-process($pt_i$).

$$\text{loc-process}(pt_i) \overset{\text{def}}{=} \text{commit}(pt_i) + (\text{abort}(pt_i) \wedge \text{replaceable}(pt_i)) + (\text{abort}(pt_i) \wedge \neg\ \text{replaceable}(pt_i))$$

If the plan, $pt_i$, is committed, then Agent$_B$ will follow Execute (n, I), and check the next plan. If the plan is aborted and replaceable (i.e., associated with alternative plans, for example allocate existing pipes or new pipes), then Agent$_B$ will replace the failed plan and execute the new plan according to Execute (n, I). However, if the plan $pt_i$ is aborted and is not replaceable, then Agent$_B$ will abort the plan.

$$\begin{aligned}\text{Terminate }(n, I) \overset{\text{def}}{=} &\ \text{if } n = \text{no-of-plans then Commit } (pt_i, I)\\ &\ \textbf{else} \sum_{i=1}^{\text{no-of-}plans} \text{Agent } (\text{decision}_i)\ \mathbf{.}\\ &\ \quad \textbf{if } \text{decision}_i = \text{abort}(pt_i)\ \textbf{then}\ \overline{\text{update}}(\text{Belief}, \text{decision}_i)\ \mathbf{.}\\ &\ \text{Abort}(pt_i, I)\\ &\ \textbf{else}\ \overline{\text{update}}(\text{Belief}, \text{decision}_i)\ \mathbf{.}\ \text{Terminate } (n + 1, I)\end{aligned}$$

$\text{Agent}_B$ checks all the commit and abort decisions about all plans. If the decision is commit then it updates its belief and checks the status of next plan by following Terminate (n + 1, I). If all the decisions are commit then it takes the Commit ($pt_i$, I) action, update the beliefs and informs other agents about the commit decision, and terminates the processing, as follows.

$$\text{Commit}\,(pt_i, I) \overset{\text{def}}{=} \overline{\text{update}}\,(\text{Belief, commit}) \,.\, \text{send-ack}\,(\text{Agent}_{pi}) \,.\, 0$$

If the decision about plan, $pt_i$, is abort, then $\text{Agent}_B$ updates its belief according to the abort decision and then take the action Abort($pt_i$, I) and terminates the processing. If some of plans are committed then $\text{Agent}_B$ also needs to compensate transactions by executing those plans.

$$\text{Abort}\,(pt_i, I) \overset{\text{def}}{=} \overline{\text{update}}\,(\text{Belief, abort}) \,.\, \text{compensate}\,(pt_i, I) \,.\, \text{send-ack}(\text{Agent}_{pi}) \,.\, 0$$

The above specification describes the behaviour of a single $\text{Agent}_B$. Using CCS re- labelling function [], we can define the behaviours of multiple agents that exhibit similar characteristics to $\text{Agent}_B$.

$$\text{Agent}_i\,(i = 1\ldots n) \overset{\text{def}}{=} \text{Agent}_B\,[\text{Plan}_i / \text{Plan}, \text{Check-depend}_i / \text{Check-depend}, \ldots\ldots, \text{Abort}_i / \text{Abort}]$$

In the above, a label pair such as $\text{Plan}_i$ / Plan defines the mapping of Plan to $\text{Plan}_i$, and Check-depend to $\text{Check-depend}_i$, and so on.

### 5.2.2 Participating Agents

Initially $\text{Agent}_p$ (participating agent) receives a request from $\text{Agent}_B$ (base agent) to cooperate in performing a task (i.e., replacing a pump). $\text{Agent}_B$ executes a plan, $pt_i$, to accomplish the requested task. In the following specification it is assumed that $\text{Agent}_p$ is requested to process one plan, $pt_i$, and that $\text{Agent}_p$ does not need other agents in processing, $pt_i$. However, if $\text{Agent}_p$ is required to process more than one plan or if it needs other agents in processing the plans, then $\text{Agent}_p$ may have a similar specification as that of $\text{Agent}_B$.

In order to process $pt_i$, $\text{Agent}_p$ first updates its beliefs database with information pertaining to the plan, $pt_i$. The behaviour of $\text{Agent}_p$ is formally modelled as follow.

$$\text{Agent}'_p \overset{\text{def}}{=} \text{newRequest}(pt_i) \,.\, \text{Agent}'_p\,(pt_i)$$

$$\text{Agent}'_p\,(pt_i) \overset{\text{def}}{=} \overline{\text{update}}\,(\text{Belief}, pt_i) \,.\, \text{Execute}(pt_i, I)$$

After updating the beliefs, $\text{Agent}_p$ executed the plan, $pt_i$, by following Execute($pt_i$, I).

$$\text{Execute}(pt_i, I) \overset{\text{def}}{=} (\text{commit}(pt_i) \,.\, \text{Terminate}\,(pt_i, I)) + (\text{abort}(pt_i) \wedge \text{replaceable}(pt_i)) \,.\, \text{Execute}(pt_i, I) + (\text{abort}(pt_i) \wedge \neg\,\text{replaceable}(pt_i)) \,.\, \text{Terminate}\,(pt_i, I)$$

If the plan $pt_i$ is committed, then $\text{Agent}_p$ will follow Terminate ($pt_i$, I). If the plan is aborted and replaceable, then $\text{Agent}_p$ will replace the failed plan and execute the new plan according to Execute($pt_i$, I). However, if the plan $pt_i$ is aborted and is not replaceable, then $Agent_p$ will terminate the plan according to Terminate ($pt_i$, I).

$$\text{Terminate}\ (pt_i, I) \overset{def}{=} \text{Commit}\ (pt_i, I) + \text{Abort}\ (pt_i, I)$$

$$\text{Commit}\ (pt_i, I) \overset{def}{=} \overline{\text{update}}\,(\text{Belief, commit})\ .\ \text{send-ack}\ (Agent_B)\ .\ 0$$

If decision about plan, $pt_i$, is abort, then $Agent_p$ updates its belief according to the abort decision, informs $Agent_B$ of the decision, and terminates the processing, as follows.

$$\text{Abort}\ (pt_i, I) \overset{def}{=} \overline{\text{update}}\,(\text{Belief, abort})\ .\ \text{send-ack}(Agent_B)\ .\ 0$$

The above specification describes the behaviour of a single participating $Agent_p$. Using CCS re-labelling function [], we can define the behaviours of multiple agents that exhibit similar characteristics to $Agent_p$.

$$Agent_{pi}\ (i = 1\ldots n) \overset{def}{=} Agent_p\ [Execute_i / Execute, Terminate_i / Terminate, Commit_i / Commit, Abort_i / Abort]$$

A label pair such as $Execute_i$ / Execute defines the mapping of Execute to $Execute_i$, and Terminate to $Terminate_i$, and so on.

In the above, we have formally specified the multi-agent transactions of the proposed model. The formal specification of the multi-agent transaction system is important, because of the complexity of the environment and the nature of interactions among various design systems. Interesting feature of the formalisation is that it guarantees the correct functionality of a system and also facilitates the implementation and verification of the system.

## 6. Conclusions and Future Work

In this paper we have discussed the issue of transaction management in distributed (possibly Web-based) design activities. We have highlighted the fact that the transactions associated with design activities are highly dynamic, cooperative, long-lived, interactive, and non-prescriptive. Moreover, these activities require fault tolerance as well as consistent and concurrent access to shared resources (e.g., databases, Web servers). These characteristics require that transactions be dynamically managed according to the extended transaction models (ETM) that suit design activitivities. We have therefore proposed a novel multi-agent transaction model, which is based on ETM and multi-agent technologies. The new model is formally defined, which provides a framework within which transactions of the Web-based design activities can be modelled according to different ETM that suit the needs of design activities.

Formal specification is crucial to model the complex transactions of Web design activities. Formal specifications also facilitate the implementation and verification of the proposed model. This model is believed to improve concurrency and fault tolerance, facilitate interaction and co-operation of participating design systems within the Web environment.

Currently we are working on the implementation of the proposed model. In order to fully evaluate the potential contributions of our model, an industrial case study is required, which is a part of the future research.

## 7. References

[1] A. Biliris, S. Dar, N. Gehani, H.V. Jagadish, K. Ramamritham "Asset: A System for Supporting Extended Transactions" *ACM SIGMOD*, May 1994, 44-54.

[2] Q.Chen, U. Dayal "Multi-agent Cooperative Transactions for E-Commerce" *Conf. on Cooperative Information Systems*, 2000.

[3] Chao, K-M, Norman, P, Anane, R., James, A. " An agent based approach for Engineering Design" *Journal of Computers in Industry*, Vol 48, No 1, 2002, pp 17-28.

[4] Sylvanus A. Ehikioya. "An Agent-based System for Distributed Transactions: A Model for Internet-based Transactions" *IEEE Canadian Conf. on Electrical and Computer Engg.*, Edmonton, Alberta, Canada, May, 1999

[5] A.K.Elmagarmid "Database Transaction Models for Advanced Applications" *Morgan Kaufman, 1992.*

[6] G.E. Kaiser "Cooperative Transactions for Multi-User Environments" *in Won Kim (ed.), Modern Database Systems: The Object Moodel, Interoperability and Beyond*, ACM Press, 1994, 409-433

[7] M. Mock, M. Gergeleit, E. Nett "Cooperative Concurrency Control on the Web" *Proc. of 5th IEEE Workshop on Future Trends of Distributed Computing System*, Tunis, October 1997.

[8] K. Nagi "Transactional Agents: A Robust Approach for Scheduling Orders in a Competitive Just-in-Time Manufacturing Environment" *Proc. of Workshop on MAS in Logistic and Economical Perspectives of Agents on Conceptulisation*, Germany, Sept., 1999

[9] E. Pitoura "Transaction-based Coordination of Software Agents" *9th Int. Conf., DEXA*, 1998

[10] C. Pu, G.E. Kaiser, N. Hutchinson "Split Transactions for Open-Ended Activities" *In proceeding of 14th VLDB Conference*, 1988.

[11] H. Ramampiaro, M. Divitini, S.A. Petersen "Agent-based Groupware: Challenges for Cooperative Transaction Models" *Proc. of the International Process Technology Workshop* (IPTW), Grenoble, Sept., 1999.

[12] Rao, S. A., & Georgeff. M. P. "BDI Agents: From Theory to Practice" *Proc. of 1st international Conference on Multiple Agent System*, 1995

[13] W. Shen "Distributed Manufacturing Scheduling Using Intelligent Agents" *IEEE Intelligent Systems*, Vol. 17, No. 1, 2002, pp 80-94,

[14] M. Younas, B. Eaglestone, R. Holton "A Review of Multidatabase Transactions on the Web: From the ACID to the SACRED" *Proc. of British National Conference on Databases (BNCOD)*, Exeter, UK, Springer, LNCS, 2000

[15] J.Yang, G.E.Kaiser, "JPernLite: An Extensible Transaction Server for the World Wide Web" *IEEE Transaction on Knowledge & Data Engineering*, 1999 (639-657)

[16] P.K. Chrysanthis, K. Ramamritham "Synthesis of Extended Transaction Models using ACTA" *ACM Transaction on Database Systems*, Vol. 19, No. 3, September 1994, Pages 450-491

[17] Yamine Ait-Ameur "*Cooperation of Formal Methods in an Engineering Based Software Development Process*" Processing of 2$^{nd}$ International Conference on Integrated Formal Methods, Dagstuhl Castle, Germany, November, 2000

[18] M. Younas, B. Eaglestone, R. Holton "A Formal Treatment of the SACReD Protocol for Multidatabase Web Transactions" 13th International Conference on Database and Expert Systems Applications (DEXA), September 2000, Greenwich, London, Springer LNCS, 899-908

[19] F. Lopez y Lopez, M. Luck, and M. d'Inverno. A framework for norm-based inter-agent dependence. In Proceedings of The Third Mexican International Conference on Computer Science, pages 31-40. SMCC-INEGI, 2001.

[20] M. Luck and M. D'Inverno, A conceptual framework for agent definition and development. The Computer Journal, pages 1-20, 44(1), Oxford, UK, 2001, Oxford University Press.

[21] Rubin Milner “*Communication and Concurrency*” C.A.R. Hoare Series Editor, Prentice Hall, International Series in Computer Science, 1989, ISBN 0-13-114984-9

[22] Glenn Bruns “*Distributed Systems Analysis with CCS*” C.A.R. Hoare Series Editor, Prentice Hall, International Series in Computer Science, 1997, ISBN 0-13-398389-7

[23] S. Conrad, B. Eaglestone, W. Hasselbring, et al. "*Research Issues in Federated Database Systems: Report of EFDBS'97 Workshop*" SIGMOD Vol. 26, No. 4, December 1997, 54-56

[24] A.P. Sheth, J.A. Larson “*Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*” ACM Computing Surveys, Vol. 22, No. 3, September 1990

[25] M. Schroeder “Will Concurrency Theory help verifying Multi-Agent Systems- A Case Study” Workshop on Multi-Agent Systems: Theory and Applications (MASTA97), October, 1997, Portugal.

[26] D. Xu, J. Yin, Y. Deng, J. Ding, " A Formal Architectural Model for Logical Agent Mobility" *IEEE Transactions on Software Engineering*, Vol. 29, No.1, January 2003.