ANALYSIS, DETECTION POLICY, AND PERFORMANCE MEASURES OF DETECTING TASK PLANNING ERRORS AND CONFLICTS

Chao-Lung Yang, Shimon Y. Nof

PRISM Research Memorandum No. 2004-P2
August, 2004

PRISM Lab/Purdue

PRISM Center
Production, Robotics, and Integration Software for Mfg. & Management

“Knowledge through information; Wisdom through collaboration”

This research is supported in part by GM R&D Center.

Juan D. Velasquez
Research Series Coordinator
PRISM Center
School of Industrial Engineering
Grissom Hall
315 N. Grant Street
Purdue University
West Lafayette, IN 47907-2023
jvelasqu@exchangepurdue.edu

Analysis, Detection Policy, and Performance Measures of Detecting Task Planning Errors and Conflicts

Chao-Lung Yang, Shimon Y. Nof

Executive Summary

In this research, we analyze the impact and severity of conflicts and errors that occur during task planning through collaboration. Taxonomy of conflict and error problems and alternative detection methods are categorized to review the different types of conflict and error situations. Three techniques are developed and applied for the impact analysis. First, a performance measure, "detect-ability", is defined to evaluate the detection potential of given detection methods. This measure and its analysis can provide the necessary decision-making information for the design of conflict/error detection. In addition, a cost-damage analysis based on available conflict/error knowledge is also applied to help decide on the optimum detection policy. Third, a viability measure is adapted for detection protocols and agents, and is applied as a measure of impact on the company to examine the detection policy and detection performance for collaborative task planners. The paper describes these three techniques, and experimental results in using them. The experiments indicate clearly that a detection policy associated with the cost-damage analysis method can maintain significantly higher viability of the collaborating planners, hence, positive total impact. This result is important, because it points to specific design guidelines for CEDM (Conflict and Error Detection Management) systems.

Acknowledgement

This research has been developed with the PRISM Center support, and support from GM R&D Center Project on "Design of Active Middleware for Error and Conflict Detection (2004)". The paper is based in part on C-L. Yang's M.S.I.E. thesis completed in August 2004.

1. Introduction

Conflicts and errors (CE) are unavoidable in distributed task planning which is performed in collaborative environments. The reason is that even if procedures for collaborative planning are well defined and well understood, each collaboration participant has his/her own expertise, local resource management, and project-based schedule. Any error caused by human lack of knowledge, or miscommunication between co-workers cannot be ignored. In order to detect the CE problems and decide whether we can afford to neglect them or must resolve them, understanding and analyzing the impact of CE problems are necessary.

The different participants in a distributed collaborative environment might have inconsistent viewpoints on the same task plan. For example, the contractor and subcontractor may have different understanding of the same task contract. These inconsistent interpretations can be treated as errors. It might be a misunderstanding of task specification by team members, or mis-estimation of task requirements, e.g., task pricing, by the task proposer. If such errors occur, or are not recognized before the task execution, the resulting error can lead to further severe consequences; e.g., integration difficulty, wrong direction of design, and unnecessary conflicts and subsequent errors. Because the cost (damage) of different or mismatched perceptions for the same task can be relatively large, detection of these errors must be achieved before executing the planned tasks.

Certain events are similar to errors and are not caused by misunderstanding, yet they may be inevitable, and may lead to errors. For example, a system may suddenly shut down because of power failure or facility breakdown. Wrong procedures followed by workers in response to such events can cause further failures, or mistakes. Usually, these types of errors are difficult to prevent, although corresponding emergency procedures can be pre-planned and trained. Detection by monitoring and inspection techniques, including sensors and validation tests, can be applied for these unexpected errors. The estimated costs of detection methods and anticipated avoidable damages, however, must be well understood before investing and applying detection techniques.

In order to investigate the CE problems in collaborative planning environments, and develop applicable detection information mechanisms and measures, two research problems have been defined:

a. ***How can the performance ability of a detection system be defined and evaluated?***

   The ability to detect a possible CE problem is important for a Co-U (Collaborating Unit participating in a Co-net, a Coordination Network) to process e-Work effectively [Nof, 2003]. Since only the CE problem can be detected, a CE resolution procedure can be launched to prevent/resolve the potential problem and damages. Detection methods with higher CE detect-ability can provide a Co-U with relatively more competence to prevent damages caused by CE events. Furthermore, a Co-U has to consider the cost issue when it performs CE detection. Applying detection methods must cost a Co-U both resources and time. It is necessary to be able to evaluate a detection method's ability by considering cost and time issues. When a Co-U can apply high detect-ability methods, it implies that this Co-U has a qualification to avoid damages caused by CE, and the detection performance can be better assured.

b. ***How can the viability measure of an independent entity be used to improve the decision-making of a detection policy?***

Once a collaborating entity decides to accept (and execute) a task request, how to perform the detection process effectively and determine the detection policy are important issues for this entity. The detection policy is a set of decisions that determine which kind of CE problems the entity wants to detect and which detection method should be applied to detect the particular CE problem. An entity can detect all CE problems by applying appropriate detection methods regardless of the detection cost, or detect selective CE problems by considering the detection cost-effectiveness. In a distributed environment, an individual entity will make independent decisions based on CE detection policy to reduce the damage of CE by paying the detection cost. Using the proper detection policy, the viability of an independent entity should be improved. Following Huang and Nof (2000), a generalized viability function can be defined to evaluate the performance of a detection policy.

The report is organized as follows: In Section 2, we briefly review some research background. In Section 3, we define the conflict and error in task planning collaborative environments. Based on that definition, we categorize the CE problems and analyze the detection methods. Section 4 describes the Detection Policy Evaluation Mechanism (DPEM) that is to evaluate detection methods by considering cost-effectiveness. Detect-ability and detection method applicability factor (κ) of a detection method are also defined. In Section 5, an experiment that applies the previously developed DPEM to compare two detection policies is addressed. Finally, Section 6 presents conclusions and recommendations.

2. Brief Research Background

2.1 The Collaborative Environment

Cooperation Unit (Co-U) -- An autonomous working unit that performs tasks to achieve its goals and coordinates with other Co-Us to accomplish the common goal of a set of Co-Us. They can be human, machine, workstation, enterprise, etc., depending on the scope of collaborative environment. A Co-U can be represented by five-tuple: G, T, A, R, and S:

$$Co-U=\{G,T,A,R,S\} \tag{1}$$

where G: Co-U's goal, e.g., profit or survival;
T: Set of tasks that Co-U must perform to achieve its goal (G);
A: Set of activities that Co-U runs in order to accomplish tasks (T);
R: Set of resources a Co-U consumes while running the activities (A);
S: Set of states presenting Co-U's current situation of G, T, A, and R.

Each Co-U performs the tasks (T) by executing the activities (A) and using limited resources (R) to achieve the goal (G). The goal of a Co-U can be defined by itself or assigned by its supervisor. A Co-U may receive tasks from other Co-Us, assign tasks to other Co-Us, or define its tasks for fulfilling its local goals. The activities are practical actions or procedures that are needed to complete tasks. In order to execute activities, each Co-U must consume its own or shared resources.

When a Co-U performs its tasks, the state (*S*) of the Co-U's current situation is changing; e.g., operation result, semi-finished good, semi-design, resource utilization, budget, viability, performance, and so on. The state (*S*) is used to describe the current situation of goal (*G*), tasks (*T*), activities (*A*), and resources (*R*).

Coordination Network (Co-net) -- was defined by Anussornnitisarn (2003). Co-net is a network that enables a collaborating process among a group of Co-Us such as agents, enterprises, and operators to achieve a common goal. Each Co-U within the Co-net exchanges information to achieve its individual tasks (*T*) and fulfill the common goal of all members. Based on Anussornnitisarn's definition, we extend the definition of Co-net by adding common-goal and resource components.

$$\text{Co-net} = \{\pi, \upsilon, \tau, \alpha, \lambda, \sigma\} \tag{2}$$

where π: Set of autonomous Co-Us in Co-net ($Co-U \in \pi$);
υ: Set of common goals that should be achieved in Co-net; *G* is a subset of τ. ($G \in \upsilon$);
τ: Set of tasks that can be processed in Co-net; *T* is a subset of τ. ($T \in \tau$);
α: Set of activities for each participant (π); *A* is a subset of α. ($A \in \alpha$);
λ: Set of resources that can be used in Co-net; *R* is a subset of α. ($R \in \lambda$);
σ: Set of coordination mechanisms that are used by each Co-U (π).

The Co-net is a whole view of operation working space. It might be an assembly line, workflow system, supply-chain system, design teamwork, or any kind of collaborative environment. Many elements comprise a Co-net, for instance, people, devices, resource, capital, social negotiation, information system, etc. All elements of a Co-U such as goals, tasks, activities, resources, are subsets of a Co-net's components, respectively.

The Co-U and Co-net define the scope of a collaborative environment. A Co-U models a participant in collaborative activities. It can be a worker, team, or company and is a basic unit that interacts with other Co-Us. A Co-net is a combination of many Co-Us. Each Co-net has its own goal (common goal) that should be achieved by involved Co-Us. Each Co-U communicates with others by certain coordination mechanisms (protocols). The tasks, activities, resources of a Co-net are preformed, executed, and shared among Co-Us, respectively.

2.2 Exception Handling

Workflow management systems (WfMSs) are increasingly being deployed to deliver e-business transactions across organizational boundaries. Developing mechanisms that allow the workflow system continue processing even if failures occur is crucial [Hagen and Alonso, 2000]. To ensure high service quality in such transactions, exception-handling schemes are needed. Klein and Dellarocas (2000) define exception as "any deviation from an ideal collaborative process that used the available resources to achieve the task requirement in an optimal way". An exception can thus include errors in performing a task or communicating results between agents, deficient response to changes in tasks or resources, missed opportunities, and so on. The exception can also include conflict that arises due to inconsistency between the sub-goal or sub-task that is performed by different participants in workflow execution. Similar definition of exception in Luo et al.'s work is that "an exception is the deviation such as system malfunctions due to failure of physical components or change in business environment" [Luo el at. 2000]. Because of anticipated or unanticipated situations, the deviations (exceptions) of those workflow processes from their specifications are unavoidable. Based on the discussion above, in our work, we treat the exception as a combination of error and conflict.

Klein and Dellarocas proposed a hierarchy taxonomy business process repository to identify the exception in a workflow system [Klein 2000, Klein and Dellarocas 2000]. It involved development and evaluation of workflow for handling the multi-agent exception that happens in a workflow environment. This centralized knowledge base can provide information to a workflow designer and human handler to recognize the possible exception problem and further resolve it for a specific business process. Accumulating exception handling experience for future exception handling is a major idea of their work.

Every exception type in the taxonomy defined by Klein and Dellarocas has an associated knowledge base entry that gives its definition. For each exception type, the workflow designer can then decide how to extend the workflow models to prepare for anticipating or detecting the exception. Pointers to the "exception detection" process templates in the repository that specify how to detect the symptoms manifested by that exception type are included in each exception type. Templates perform as "sentinels" that check for symptoms of actual or imminent exception. In our work, we apply this taxonomy concept to develop CE knowledge base that can store any information about the CE problems and possible detection methods.

Luo et al. (2000, 2003) identify exception-handling techniques that support conflict resolution in cross-organizational settings. In particular, the authors classify four abnormal situations in cross-organization processes: 1. a contract cannot be fulfilled; 2. a contract may be compromised; 3. a contract needs to be modified; 4. a contract must be terminated before it expired. All these situations can be treated as conflict problems because they violate predefined goals among collaborative participants. In their work, a three-dimensional analysis of exception in a workflow system is addressed. Three characteristics: known, detectable, resolvable are used to describe or analyze each exception. Especially in detectable characteristics, the authors mention that detection of an exception depends on the system's ability. If a system can recognize or detect the exception, it may be possible to derive capable exception handlers to solve the problem. Consequently, the detect-ability of a system will determine the performance of the detection process. In our research, we define the detect-ability measure to represent the detection competence of a Co-U.

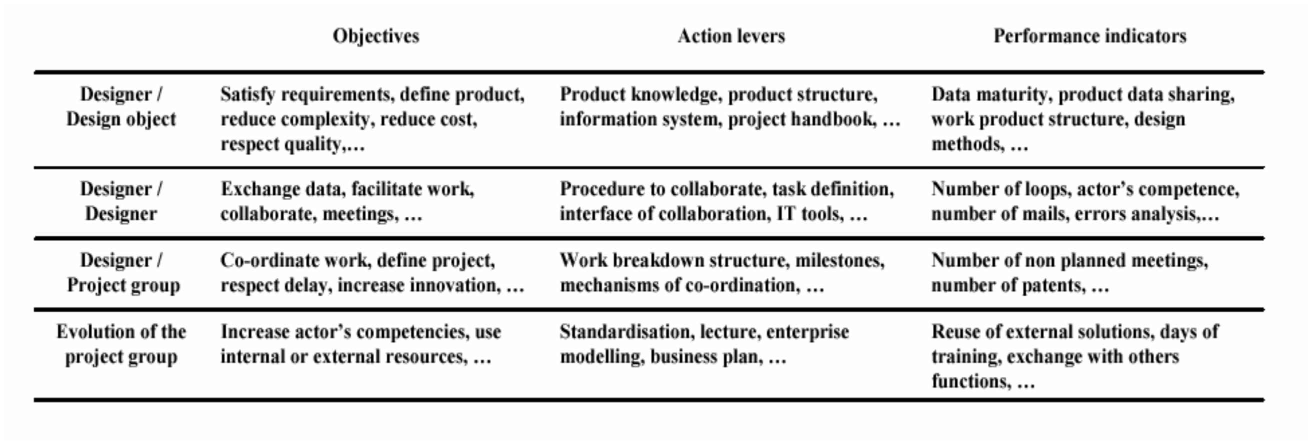
2.3 Performance Evaluation

Taxonomy of collaborative design that is relevant to our research was proposed by Ostergaard and summers (2003). It considers human, organizational and procedural aspects, and includes six attributes: Team composition; Communication; Distribution; Design approach; Information; and Nature of the problem. Based on this taxonomy, several parameters were defined to describe the collaborative design environment.

Collaborative performance evaluation is an important issue in a collaborative planning environment. Robin et al. (2004) proposed the GRAI model (Graphs with

Results and Actions Inter-related) of the design environment, to evaluate the design process performance. They focus on evaluating performance with a specific view on the design actors. Based on the design environment, several performance indicators can be defined to evaluate the performance of collaborative process, addressing the actor and the design object; the actor interactions with another actor; the actor and the group; and evolution of a group of actors (Table 1). Definition of the design environment and collaborative design process provide a clear view of the interactions between collaborative participants: human, design team, objectives, actions, and performance indicators. In different interactions, various aspect of design teamwork can be investigated. However, all these performance measures assume a conflict-and-error-free environment. Once a CE problem occurs, how to measure the impacts on an enterprise from having the ability of CE detection for prevention and resolution has not yet been addressed. In this research, we address the CE problems and develop and adapt several measures and methods to evaluate a Co-U's detection performance.

Table 1: Interactions between objectives, action levers, and performance indicators -- evaluation focused on actors (Robin et al., 2004)



	Objectives	Action levers	Performance indicators
Designer / Design object	Satisfy requirements, define product, reduce complexity, reduce cost, respect quality,…	Product knowledge, product structure, information system, project handbook, …	Data maturity, product data sharing, work product structure, design methods, …
Designer / Designer	Exchange data, facilitate work, collaborate, meetings, …	Procedure to collaborate, task definition, interface of collaboration, IT tools, …	Number of loops, actor's competence, number of mails, errors analysis,…
Designer / Project group	Co-ordinate work, define project, respect delay, increase innovation, …	Work breakdown structure, milestones, mechanisms of co-ordination, …	Number of non planned meetings, number of patents, …
Evolution of the project group	Increase actor's competencies, use internal or external resources, …	Standardisation, lecture, enterprise modelling, business plan, …	Reuse of external solutions, days of training, exchange with others functions, …

2.4 Viability

According to Huang and Nof (2000), the viability *Vi* of an autonomous agent is characterized by Eq. 3: s_i represents a static viability that is given and unchangeable when an agent is designed; $d_i(t)$ is a dynamic viability at time *t*. A static viability consists of two trade-off variables: agent's resource capability, and operating cost of agent. Moreover, a dynamic viability is a performance (reward) of an individual agent over a period of time. A general viability can be measured as a combination of these static and dynamic viability measures. Huang et al. (2000) concluded that viability is not only a measure of an autonomous system, such as an agent based manufacturing system (ABMS), but also an important characteristic of any autonomous system. We can use this viability to evaluate the functional health or performance of an autonomous system. In this research, the static viability will not be considered because the conflict/error situation is always triggered dynamically.

$$Vi = \{s_i, d_i(t)\} \quad (3)$$

where s_i is the static viability of agent *i*;
$d_i(t)$ is the dynamic viability of agent *i* at time *t*.

Definition of dynamic viability of an agent

The dynamic viability $d_i(t)$ is defined below.

$$d_i(t) = \frac{W_i(t)}{w_i} \bigg/ \overset{n}{\underset{j=1}{Max}}\left(\frac{W_j(t)}{w_j}\right) \qquad (4)$$

where $W_i(t)$is accumulated rewards of agent *i* at time *t*;

$\frac{W_i(t)}{w_i}$ shows how long agent *i* could maintain its operation by using the accumulated rewards;

$\overset{n}{\underset{j=1}{Max}}\left(\frac{W_j(t)}{w_j}\right)$ is the largest $\frac{W_i(t)}{w_i}$ in an system.

Detailed background and specification of viability can be found in [Huang and Nof, 2000]. In this research, we use dynamic viability to evaluate the Co-U's performance and health. The received reward from performing a task can support the Co-U to continue its operation. How to eliminate the damage on rewards by applying a detection process is a core objective of the detection model.

3. Conflict and Error Problems in Collaborative Planning

Before addressing conflict and error problems in collaborative planning, we make some assumptions to describe the research background:

1. In a collaborative environment, Co-Us are willing to detect and solve the problem of planning errors and conflicts together. It means that Co-Us will want to share their current information and knowledge on CE detection.

2. Collaborative members have a common goal to achieve. For example, design teams with different expertise want to design a product together. Each organization of an enterprise makes concerted efforts for achieving the maximum benefit for the enterprise.

Based on these assumptions, we can conclude that two types of general collaborative conflicts, time-conflict and integration-conflict, may occur in a collaborative planning environment.

Time Conflict (TC) -- Time-conflict is caused by the different schedules or plans of autonomous organizations. There are two types of time-conflict: ***resource conflict*** and ***schedule conflict***. Resource conflict may occur if Co-Us share common resources but have a different plan or schedule on using resources. The overlap of occupying these resources can cause a resource conflict. The common resource might be a facility, location, labor power, funds, service, network bandwidth, and so on.

Even when Co-Us do not share common resources, schedule conflict may still occur. In a collaborative environment, every Co-U has its own task schedule based on its current situations and task requirements. The different processing times and finished times of Co-Us may affect other Co-Us' subsequent tasks. This interdependence can cause schedule conflicts when merging all the schedules of collaborative Co-Us together.

These two types of conflicts may happen simultaneously and be correlated if common resources are used. Since a task finish time may depend on its resource-utilizing arrangement, a schedule conflict may be caused by a resource conflict.

Integration Conflict (IC) -- In a collaborative environment, tasks finished by different Co-Us might need to be merged together. The integration conflict happens when results of tasks interfere with each other, instead of being merge-able. This kind of conflict is common in concurrent engineering, or design teamwork. When different design teams with different expertise plan to co-design production tasks together, the varied design considerations may cause an integration conflict.

Integration conflicts may also occur when different Co-Us wish to "assemble" their tasks or services. When different Co-Us wish to integrate their task results to become a single production or service unit, any inconsistency between their outcomes and expected results may cause an integration conflict. For example, in an assembly line, a work station is in charge of assembling the components produced by preceding work stations. The integration conflict occurs if the dimensions of assembled components supplied by different sources do not fit within specified assembly tolerances or assembly standards.

Because of task interdependence among Co-Us, time and integration conflicts may be widespread within a Co-net. All participants must detect or at least share information about detected CE problems once one of them has detected unresolved or ill-resolved CE problems. How to detect CE and inform other involved and influenced Co-Us is an important issue in CE detection. In the next section, we discuss task processing CE problems in a task planning environment that are the focus of this report.

3.1 Conflicts / Errors in Task Planning

Categories of specific CE situations that may happen in a collaborative planning environment are explained. In this research, detecting CE problems that occur after Co-Us accept tasks is our main interest.

Taxonomy of CE in Task Planning

Applying coordination protocols for distributed environments, such as the Resource Allocation Protocol (RAP) developed by Anussornnitisarn and Nof (2003), or other task assignment protocols, such as Contract Net protocol [Smith, 1980] and "citizen" contract net [Dellarocas and Klein, 2000], a task can be rationally assigned in a Co-net to specific Co-Us. Then, a task plan can be established and each Co-U performs its task to achieve its goal. No matter which kind of task allocation protocol the Co-net uses, the major role of each Co-U will be either *contractor* or *proposer*. Figure 1 shows, as an example, a simple version of the Contract Net protocol.

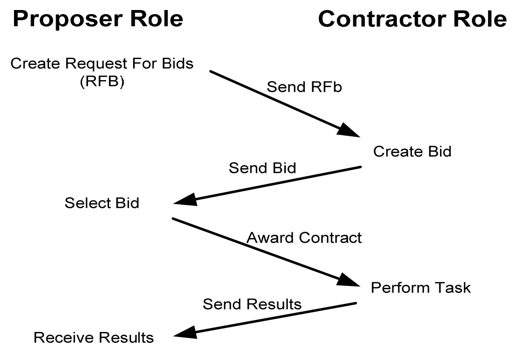


Figure 1: A simple version of a coordination protocol (e.g., Contract Net protocol) (Modified from Dellarocas and Klein, 2000)

Conflicts and errors, however, may happen after each Co-U receives the task. For example, a task proposer might find out mis-estimates of a task after its contractors receive the task specification and begin to execute it. In this case, if the mis-estimates are not detected, or a recovery process is not activated, the conflict between the proposer's expected task results (proposer's original goal) and contractor's finished results (contractor's goal) will eventually occur. Then, both of the proposer and contractor cannot be comfortable with this conflict situation. This kind of CE is highly common in practice. We strive to prevent this kind of CE. Other possible CE situations will be discussed in the next section based on the roles of the task proposer and contractor. Again, in this research, we focus on detecting these CE situations in a Co-net after tasks have been created and assigned.

3.2 Conflicts / errors for Task Proposer and Contractor

Conflict type I: Wrong task specification sent by the task proposer (information conflict)

This type of conflict happens when there is an oversight of task specification by the task proposer, or the proposer provides wrong task information to the contractor. It means that the wrong task specification is not realized during the task allocation "contractual" stage. This conflict type is caused by a proposer's task specification being inconsistent with the contractor's understanding. Three types of errors may lead to this conflict type:

a. *Improper goal of the proposer*

   The proposer defines a wrong task specification based on an improper goal.

b. *Task specification oversight*

   Before the task allocation stage, the proposer defines the wrong task specification and makes a decision based on it.

c. *Mistaken task specification sent out*

   The task proposer sends, by mistake, wrong the task specification to the contractor.

In this case, there are two possible detection situations: One is that a proposer realizes the mistake on task specification after the task has been accepted, but not yet finished. Another situation is that the proposer becomes aware of this problem only after the contractor completes the task.

a. *After a task is assigned and before this task is finished*

   This conflict is detected by a proposer when an error is found before a contractor finishes the task.

b. *After the task is finished*

   Before the task is finished, both the contractor and proposer do not detect this task conflict. After the contractor finishes the task, the result is not consistent with the proposer's expectation.

If the proposer can detect this type of conflict in the very beginning, the damage of Conflict type I may not be so high, relatively. If this conflict is detected after the task is completed, the harm can be significant for both the proposer and contractor.

Conflict type II: The contractor cannot finish tasks

This type of conflict occurs when a contractor cannot finish an assigned task. The conflict, in this case, is that the contractor cannot finish the task to fit the proposer's expected result. This type of conflict/error might be caused by several possible errors. They are:

a. *Resource unavailable and unrecoverable before deadline*

   Unexpected events or nature mishaps, e.g., power blackout, earthquake, human or facility fatigue, might cause this type of error.

b. *Mis-estimated utilization*

   After a contractor receives a task offer, a mis-estimate on the resource appraisal is realized because required resources are no longer available. For example, the contractor neglects the price fluctuation of raw material and the budget is short to prepare the adequate amount of material.

c. *Contractor misunderstood ability or capacity*

   This error is similar to resource mis-estimate, but it is caused by the mis-estimate on the ability or capacity appraisal. For example, the contractor overestimates its ability for received tasks and finally cannot fulfill the task requirements.

The damage from Conflict type II depends on when this type of conflict/error is detected once it happens. If the mis-estimate situation is detected relatively very late, or sudden shutdown event happens in the very beginning, the harm will be relatively large. Usually, this type of conflict also causes a time conflict.

Conflict type III: Task has been finished but the result is faulty

This type of CE problem happens when a contractor finishes a task but the contractor or proposer discovers that the results do not fit the requirements (specifications) of the task. This conflict is detected when either the contractor or proposer realizes the task result is not consistent with the proposer's task specification. Two possible errors may cause this conflict:

a. *Contractor misunderstood task specification*

   This conflict type happens when a contractor misinterprets the task specification (although the received task specification is correct) and continues to work until the task is finished. It can occur frequently in a design project. For example, a software designer misunderstands the functional requirements of the developing automated device until delivery acceptance tests reveal the discrepancy.

b. *Wrong or negligent task activities*

   If a contractor errs by executing a task inaccurately or negligently, this type of conflict will happen. It is common in manufacturing processes. For example, using a wrong tool (an error) can cause this type of conflict.

Conflict type III is usually severe since the cost and time of performing the task are already spent, yet a wrong task-result is available. Usually, this type of conflict also causes an integration-conflict and is usually detected in the “check and accept” late stage. For example, the assembled components with wrong geometric dimensions will cause the assembly to fail. According to the discussion above, Table 2 summarizes the conflict types, possible errors causing them, and resulting damage levels.

Table 2: Conflicts after tasks have been assigned in a Co-net

Conflict Type:	**Conflict Type I** Wrong task specification sent by task proposer	**Conflict Type II** The contractor cannot finish Task	**Conflict Type III** Task is finished but result is faulty
Definition	Task specification inconsistency	A contractor cannot achieve the task to fit the proposer’s expected result	A contractor’s task result is not consistent with the proposer’s task specification
Who errs	Task proposer	Task contractor	Task contractor
Causal Error	1. Improper goal of proposer 2. Task specification oversight 3. Mistaken task specification sent out	1. Resource unavailable and unrecoverable before the deadline 2. Mis-estimated utilization 3. Contractor misunderstood ability or capacity	1. Contractor misunderstood task specification 2. Wrong or negligent task activities
When errors occurs	Before the task is assigned	After the task is assigned and before the task is finished	After the task is finished
Damage level	1. Depends on when this conflict is detected 2. If the task is finished, the damage is relatively high 3. Before the task is finished, the damage is relatively low	Depends on when this conflict is detected	Relatively high

3.3 Conflict and Error Damage Analysis

Based on the discussions regarding category of conflict/error types in the previous sections, Figure 2 briefly shows the occurrences and damage levels of different conflict types. In Figure 2, we assume for simplicity that the damage level is linear and depends on the conflict occurrence time. We can notice that the damage caused by these conflicts will be higher the later this conflict is detected.

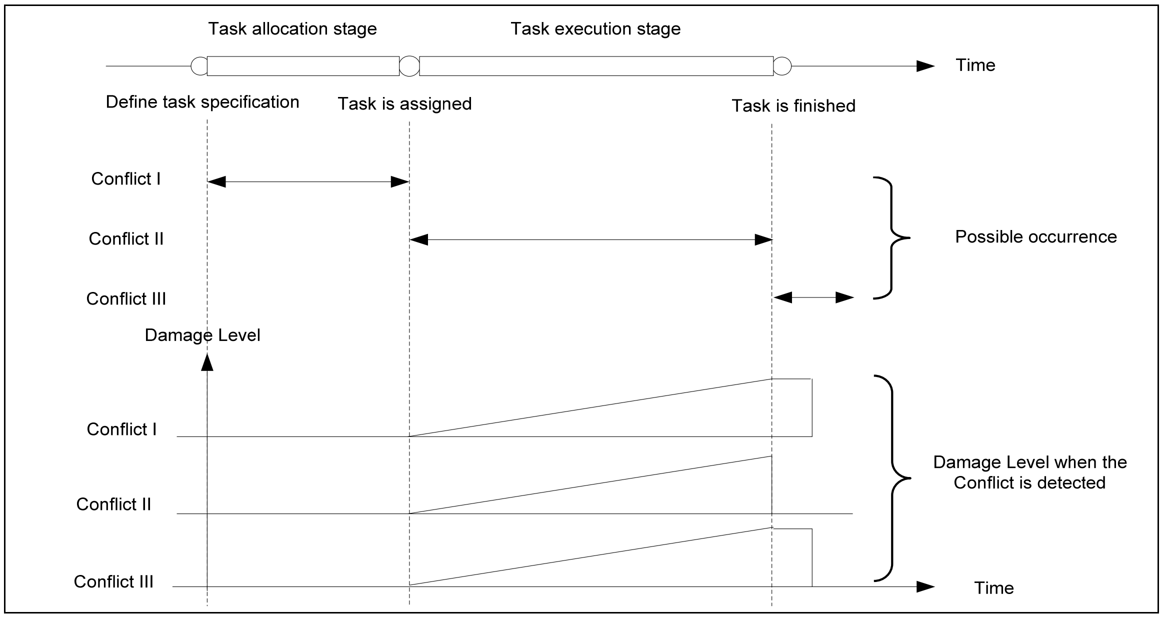


Figure 2: Possible occurrence and damage level of three conflict types

Having categorized and explained the conflicts / errors, a brief conclusion can be drawn. If we can detect conflicts and errors as early as possible, the damage of the conflict/error will be smaller, or completely prevented. Although this conclusion seems obvious, a crucial question is how to design an effective detection protocol that enables Co-Us to detect conflicts and errors. Following the CE categorization, we can next analyze the errors causing each conflict (Table 3).

Through the taxonomy of conflict and error, an inductive conclusion on detection or resolution method can be found [Klein 2000]. In this paper, after classifying the CE problems, we recognize that applying an appropriate detection method for a specific CE problem and exchanging detection information through an applicable protocol among Co-Us are crucial in CE detection. For example, if we can perform appropriate detection methods such as “double-confirm” before the contractor performs the task, the possibility of misinterpretation of task description can be decreased great deal. By using an applicable detection protocol, Co-Us can share error monitoring results with each other or warn other correlated partners about that error in advance.

Based on the previous conclusions, a detection protocol has been developed in this research to detect the task processing CE problems. Before a contractor executes a task, exchanging the perception of task specification between the contractor and proposer will be the first step in the detection protocol. In this step, the task proposer reviews the task specification sent from the contractor to check whether there is any difference on the perception of task specification. In the meantime, the contractor also checks whether the task specification sent from proposer is identical with its perception on the task. Both the proposer and contractor can also review whether the ability and resources are sufficient for performing this task. Briefly, this step of detection process is a “re-confirm” activity between the task proposer and contractor. Through this information exchange, both the proposer and contractor can reduce the misunderstanding in common goals or in offering and accepting task contracts.

Table 3: Analysis of CE causes and corresponding detection method

Error Type	Description	Error Characteristic	Detection Method	Detection Type
Conflict Type I: Wrong task specification sent by the task proposer				
Error 1	Improper goal of the proposer	Mis-estimate	Exchange task specification to re-confirm with proposer before executing task	Single time
Error 2	Task specification oversight	Mistake	Exchange task specification to re-confirm with proposer before executing task	Single time
Error 3	Mistaken task specification sent out	Mistake	Exchange task specification to re-confirm with proposer before executing task	Single time
Conflict Type II: The contractor cannot finish tasks				
Error 1	Resource unavailable and unrecoverable before deadline	Resource allocation error	Constantly check resource availability and utilization plan	Continuous
Error 2	Mis-estimated utilization	Resource allocation error	Constantly check and re-evaluate resource utilization	Continuous
Error 3	Contractor misunderstood ability, capacity	Misunderstanding	Contractor constantly re-evaluate its ability, capacity before executing task	Continuous
Conflict Type III: Task has been finished but the result is faulty				
Error 1	Contractor misunderstood task specification	Misunderstanding	Proposer re-confirms contractor's task specification perception before executing task	Single time
Error 2	Wrong or negligent task activities	Process error	Constantly monitor/verify task progress	Continuous

In terms of continuous detection, a detection agent can play the role of "sentinel" [Klein, 2000] to constantly collect information from resources, facilities, human activities, and so on. According to the collected data, the error symptom can be detected by comparing the expected state with the current state based on the collected data. A decision can be made to determine whether the error has happens (or about to happen), and which Co-U is involved with the CE event.

Once the CE event is detected, the Co-U should launch the CE recovery or resolution model (CERM) to handle the CE problem. Although CERM is beyond the scope of this paper, all CE-involved Co-Us should provide complete detection information to the CERM (or to a human operator/manager) and clearly describe the CE situation. In addition, the detection process also requires certain functions, such as decision support and learning ability, to consolidate and strengthen the CE detection.

In summary, this section studies CE problems in a task planning collaborative environment. Through studying the CE problem after tasks are assigned, we conclude that applying applicable detection methods by considering the cost of the detection method and damage of CE problem to detect the CE problem is an important issue.

4. Detection and Evaluation Methodology

In this section, we discuss the CE detection method analysis and develop the Detection Policy Evaluation Mechanism (DPEM). First, we define a CE table to store the information about the CE problem and its detection method. Then, we propose a measure, detect-ability (*ß*), to evaluate the CE detection method. Based on the detect-ability (*ß*) of detection methods, a detection policy that considers the cost and potential damage of a given CE problem is determined for a Co-U to perform its CE detection processing.

4.1 Conflict / Error Knowledge Base

The CE knowledge base is a repository, or a dynamic ("active") database of CE detection information. It can be accessed by Co-Us. A CE table is defined and maintained to store all information about the CE problems in the CE knowledge base. The CE table is a basic data unit to store certain information, such as CE index (x), CE description, CE occurrence frequency (F_x), CE detection method (dM_{xy}), detection accuracy (dA_{xy}), average detection cost (dC_{xy}), detection cycle time (dT_{xy}), possible damage (D_x), and potential CE resolution. The structure of a CE table and an illustration are shown in Table 4.

Table 4: The CE table of a task *T*

Index of Co-U	
CE index (*x*)	The unique index of a CE problem
CE description	The description of this CE
CE occurrence frequency (F_x)	F_x is the occurrence frequency of a CE event. It is the number of occurred CE problems per time unit. For example, F_x might be 0.01/hour. Higher frequency means this CE has a higher probability to occur.
Damage (D_x)	The estimate of total possible damage of a CE problem
Detection methods (dM_x) Detection method (dM_{xy}) $\forall dM_{xy} : dM_{xy} \in dM_x$	dM_x is a set of applicable detection methods for CE *x* dM_{xy} is a detection method y to detect CE *x* Storage of a set of applicable detection methods that can be used to detect CE *x*. There could be more than one useful detection method available to detect a CE problem. *y* -- the index of the detection method (*y* >= 1)
Detection accuracy (dA_{xy})	The accuracy of specific detection method y, e.g., detection rate of sensor
Average detection cost (dC_{xy})	dC_{xy} is the average detection cost per time unit of the specific detection method y to detect CE problem *x*. For instance, dC_{xy} might be \$10/second or \$600/minute, etc. Each detection method may have a different detection cost.
Detection time (dT_{xy})	dT_{xy} is the time spent on finishing a detection cycle of the specific detection method y. For example, in manufacturing, a visual sensor might need 2 seconds to finish inspection. In this case, the *dT* is 2 seconds. Each detection method may require a different detection time.
Potential CE resolution	This element describes the possible CE resolutions

A CE table is stored in a CE knowledge-base and categorized by Co-U's task (*T*). In terms of task processes, several possible CE problems might happen in a Co-U. A CE table is a systematic storage that is input by Co-Us in a Co-net. CE problems can be registered in a CE table, in which several detection methods are contained. Therefore, CE tables can be treated as common accumulating experiences of CE detection in a Co-net. The information of detection methods stored in a CE table provides the Co-U sufficient data to analyze or examine possible detection procedures when the Co-U faces an analogous CE problem.

4.2 Detection Policy Evaluation Mechanism (DPEM)

Detection Policy Examination Mechanism (DPEM) is responsible for evaluating each detection method (*dM*) regarding each possible CE problem and recommending the detection policy $(dP)_T$ for executing a task *T*. The detection policy is a guideline of how to detect a particular CE problem. According to the information stored in the CE knowledge base, DPEM can evaluate all applicable *dM* for a specific CE and recommend the best one if only one *dM* is required by comparing Detect-ability (*ß*). Then, DPEM will evaluate the cost-effectiveness of the recommended *dM* and decide whether it should be applied, based on calculation of Detection Method Applicability Factor (κ). After evaluations, DPEM recommends the effective detection policy $(dP)_T$ for executing the CE detection when the Co-U is ready to perform a task *T*.

The operation of the DPEM is illustrated in Figure 3. In the following sections, we discuss the details of this detection mechanism.

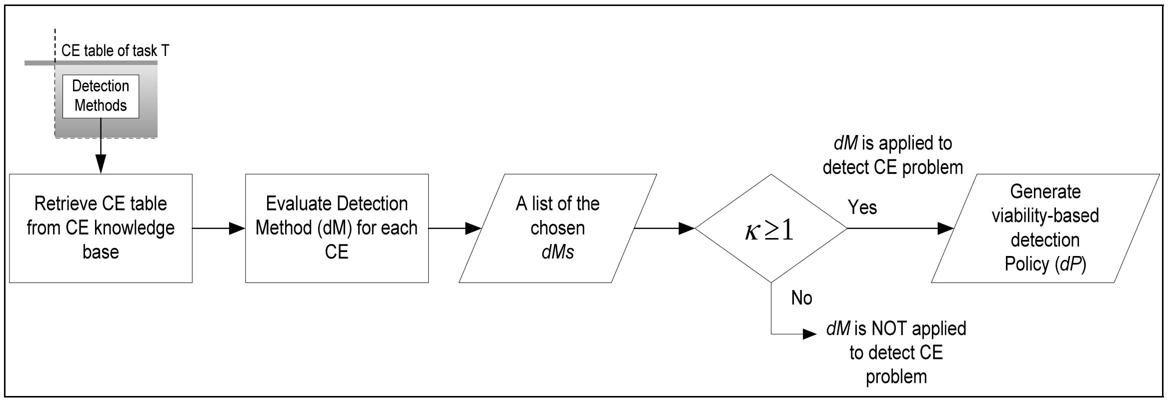

CE table of task T
Detection Methods
Retrieve CE table from CE knowledge base
Evaluate Detection Method (dM) for each CE
A list of the chosen dMs
κ≥1
dM is applied to detect CE problem
Yes
Generate viability-based detection Policy (dP)
No
dM is NOT applied to detect CE problem

Figure 3: Operation of the DPEM

Detect-ability (*ß*)

The detect-ability (*ß*) is defined as a measure to evaluate a detection method's competence in detecting CE. Based on this measure, the DPEM can select the most applicable detection methods. *ß* can be defined by using the CE information in the CE table, as follows.

$$(\beta)_{xy} = \left(\frac{dA_{xy}}{dC_{xy} \times dT_{xy}} \right)_{dM_{xy}} \tag{5}$$

where *x* is the index of a CE;

y is the index of a detection method for CE *x*;

$(\beta)_{xy}$ is the detect-ability measure of detection method y for CE *x*;

dA_{xy} is the accuracy of the detection method y for CE *x*;

dC_{xy} is the average detection cost per time unit of detection method y for CE *x*;

dT_{xy} is the detection time of detection method *y* for CE *x*.

The detect-ability $(\beta)_{xy}$ of a detection method y for CE x (*dM*$_{xy}$) can be calculated as the ratio of detection accuracy (dA_{xy}) over average detection cost (dC_{xy}) multiplied by detection cycle time (dT_{xy}). If dC_{xy} or dT_{xy} are relatively high, $(\beta)_{xy}$ will be smaller since the Co-U needs to spend more cost and/or time on the detection process. In addition, since a detection method might not reach 100% accuracy on a given CE detection, the accuracy should be considered in calculating the detect-ability measure.

During the detection process, several CE problems might occur while a Co-U executes a task *T*. Regarding each possible CE, several detection methods might be applicable. Based on this measure, if the *ß* of a detection method is relatively higher, it means that this detection method is better compared to others with lower *ß*. Therefore, the DPEM of a Co-U can evaluate and compare alternative detection methods by *ß* and select the recommended one if one detection method is required.

Detection Method applicability Factor (κ)

Once a *dM* is chosen from available detection methods, the system continue to evaluate cost-damage analysis of chosen *dM.* According to CE information stored in the CE table, we propose a decision-making logic to decide whether a particular *dM* should be applied based on cost perspective (not ability perspective). The proposed logic is addressed in the following equations.

$$(I)_{dM_x^{''}} = (1 - dA_{xy}) \times F_x \times (T_d)_x \times D_x + dC_{xy} \times (T_d)_x \tag{7}$$

where *x* is an index of a CE problem;

y is an index of a detection method;

$(T_d)_x$ is the total detection process time for detecting CE *x*;

$dM_x^{''}$ is an applicable detection method that is selected from dM_x to detect CE *x*;

$(I)_{dM_x^{''}}$ is the accumulated damage and detection cost if a $dM_x^{''}$ is applied to detect CE *x* in $(T_d)_x$;

F_x is the occurrence frequency of a CE *x* (No. of occurred CE per time unit);

D_x is damage from a CE *x*;

dA_{xy} is the accuracy of the detection method *y* for CE *x*;

dC_{xy} is the average detection cost per time unit of detection method *y* for CE *x*.

$$(II)_x = F_x \times T_{dx} \times D_x \tag{8}$$

where *x* is an index of a CE problem;

$(T_d)_x$ is the total detection process time for detecting CE *x*;

$(II)_x$ is accumulated damage if no detection method is applied to detect CE *x* in $(T_d)_x$;

F_x is the occurrence frequency of a CE *x* (No. of occurred CE per time unit);

D_x is damage from a CE *x*;

$(T_d)_x$ is the total detection process time for detecting CE *x*.

Based on Eq.s 7 and 8, $(I)_{dM_x^{''}}$ is the accumulated damage plus detection cost if a selected $dM_x^{''}$ is applied to detect CE *x*. On the other hand, $(II)_x$ is the accumulated damage if no detection method is applied to detect CE *x*. We can compare $(I)_{dM_x^{''}}$ and $(II)_x$ to decide whether $dM_x^{''}$ should be applied or not. If $(I)_{dM_x^{''}}$ is greater than $(II)_x$, we should not apply $dM_x^{''}$ to detect CE *x* because even if we spend the detection cost, the overall cost-effectiveness is worse than "do-not-detect-it". Here, we define the applicability factor (κ) of a $dM_x^{''}$ as the ratio of $(II)_x$ and $(I)_{dM_x^{''}}$ to evaluate the selected $dM_x^{''}$ according to relative cost, time, and accuracy.

$$\kappa = \frac{(II)_x}{(I)_{dM_{xy}}} = \frac{F_x \times (T_d)_x \times D_x}{(1-dA_{xy}) \times F_x \times (T_d)_x \times D_x + dC_{xy} \times (T_d)_x} = \frac{F_x \times D_x}{(1-dA_{xy}) \times F_x \times D_x + dC_{xy}}$$

if $\kappa < 1$ Do not apply $dM_x^{''}$ to detect

if $\kappa \geq 1$ Apply $dM_x^{''}$ to detect ($dM_x^{''} \rightarrow dM_x^{'}$) (9)

where κ is applicability factor of *dM_x*,

$dM_x^{''}$ is a applicable detection method that is selected from dM_x to detect CE *x*;

$dM_x^{'}$ is a detection method that is chosen from dM_x to detect CE *x* and can maintain relatively higher viability of a Co-U;

Based on κ in Eq. 9, the DPEM can decide whether it will apply $dM_x^{''}$ to detect a particular CE or not. If $\kappa < 1$, it means that the cost-effectiveness of applying this $dM_x^{''}$ is even worse than for not applying it. It also means that the viability of a Co-U will be harmed if this $dM_x^{''}$ is applied. Therefore, this $dM_x^{''}$ should not be included in the *$(dP)_T$*. On the other hand, if $\kappa \geq 1$, this $dM_x^{''}$ will be selected as $dM_x^{'}$ and be included in *$(dP)_T$* for performing detection.

Viability-based detection policy *(dP)$_T$*

Viability-based detection policy *(dP)$_T$* is a set of detection methods that are used to monitor a Co-U's activities after evaluation of their contribution to maintaining relatively higher dynamic viability of a Co-U. It is defined as follows:

$$(dP)_T = \{dM_1^{'}, dM_2^{'}, .., dM_x^{'}, ..., dM_N^{'}\} \mid \forall dM_x^{'} : dM_x^{'} \in dM_x \tag{6}$$

where *T* is a task performed by a Co-U;
N is the total number of CE problems;
x is the index of a CE;
(dP)$_T$ is a viability-based detection policy for performing task *T*;
dM$_x$ is a set of detection methods for CE x;
$dM_x^{'}$ is a detection method that is selected from *dM$_x$* to detect CE *x* and that can maintain relatively higher viability of a Co-U.

In general, detecting all possible CE problems is a usual detection policy. The purpose of this policy is to detect any possible CE problems to prevent damage caused by CE problems. In order to execute this policy, for each possible CE, a Co-U must apply at lease one detection method. However, since this detection policy does not consider the correlation of average detection cost and damage level of a CE, it might lead to a Co-U spending tremendous cost to detect the CE problems which cause slight or negligible damages in an extreme case. In other words, in this case, the cost-effectiveness of applying "detecting-all-CE-problems" policy will be low and wasteful. That low cost-effectiveness detection policy will harm the viability of a Co-U. Therefore, after the DPEM selects the applicable *dM*, the DPEM must decide whether the recommended *dM* should indeed be applied or not by cost-damage analysis.

The average detection cost (*dC*) of a detection method is the average expense that each Co-U has to spend per time unit in detecting CE. The *dC* is a part of operation cost that Co-U needs for maintaining its operations. If the *dC* is very high and the CE problem appears rarely, it means that a Co-U has to spend more money or energy on continually detecting the CE. It will reduce the benefit or profit from performing a task.

In addition, the potential damage caused by a CE problem is also an important consideration for Co-U to apply CE detection methods. The damage might be the profit lost, time wasted or reputation loss in case the CE is not detected. In a collaborative environment, a Co-U with frequent CE problems might not be a good partner because the CE problem will reduce and harm the trust and value of partnership among Co-Us. If the damage of a CE is very high, each Co-U should try to detect that high-damage CE and eliminate that potential damage. Therefore, the trade-off between detection cost and CE's damage is a major consideration of the DPEM.

According to the measure *ß* and applicability factor κ, the DPEM can evaluate detection methods and generate a viability-based (relatively-high-cost-effectiveness) detection policy. In the next section, we design an experiment to evaluate the detection policy based on the strategy of selecting detection methods.

5. Experimental Results and Discussion

The purpose of this experiment is to examine alternative CE detection policies. Generally, detecting all possible CE problems is one of several feasible detection policies and is often the intuitive practice in manufacturing and business. However, before we automate the detection processes, it may be necessary in certain cases to consider other policies. The reason is that detecting all possible CE problems for a Co-U may cost too much and achieve relatively low overall results with CE detection. For instance, if the cost of a detection method is prohibitive while the preventable damage by applying this method is relatively low, the cost–effectiveness of applying this detection might not be acceptable. In other words, we can hypothesize that a Co-U can maintain higher viability by enhancing its cost-effectiveness with a rational, selective detection policy.

In this experiment, we use a viability measure to evaluate a Co-U's detection performance. The higher detection-based dynamic viability a Co-U can maintain when it performs CE detection, the higher cost-effectiveness of CE detection the Co-U can achieve. By considering the gains (preventable damage) and losses (detection cost), a better detection policy can be made to optimize the detection activity and maintain higher viability of a Co-U. It is assumed that a higher viability of Co-Us implies a higher total viability of the Co-net. In other words, the impact of effective detection can be summarized for the total effective impact for the company.

We compare two detection policies. One is detecting all CE problems and another is applying DPEM evaluation logic to generate the viability-based detection policy $(dP)_T$. The detection-based dynamic viability is introduced to evaluate a Co-U's performance in performing detection processes.

5.1 Experimental Assumptions

a. When a Co-U accepts a task request, all possible CE problems regarding this task can be obtained from the CE table which had been formulated in advance.

   This assumption is reasonable in a practical environment. For instance, in manufacturing, possible CE problems such as facility shut-down, operation error, schedule delay, or material shortage can be predicted based on previous experience, the result of a trial-run, the facility's availability, or risk analysis. We can assume that CE problem information; e.g., CE problem description, detection method, detection cost, and damage of a CE can be obtained from the CE table described in the Section 4.

b. If a CE is detected, the damage of the CE can be prevented.

   Once a CE is detected, the CE resolution mechanism (automated or manual) will be launched to solve the detected CE problem. Because the CE resolution mechanism is beyond the scope of this research, in this experiment, we assume that no damage is caused if a CE problem is detected.

c. The average detection cost is constant during detection.

   The average detection cost is defined as detection expense per time unit. Therefore, the total detection cost can be calculated by multiplying the average detection cost per time unit by total detection processing time. Although not all detection methods might cost a constant expense, e.g., applying a sensor or detection device, we treat the cost of applying detection methods as the continuous expense, for experimental convenience.

d. The damage (*D*) from a CE problem is significantly higher than average detection cost per time unit (*dC*).

   In general, the damage from the CE problem could be tangible or intangible, e.g., the loss of capital or time (tangible), or ruined reputation (intangible). Whether the damage is tangible or not, we assume the damage is greater than the average detection cost (cost/time). For example, the damage from a CE could be $1000 if this CE is not detected and the *dC* of applying a *dM* could be $2/hour. According to different total detection processing times, the total detection cost may be greater than total accumulated damages. Based on this concept, a better detection policy can be sought.

e. The detection time (*dT*) of a detection method is negligible.

   We assume the detection time of a detection method is relatively smaller than task operation time.

f. For experimental convenience, only one detection method is applied for each CE problem.

   In some cases, multiple detection methods might be applied to detect a particular CE problem. In this experiment, we assume only one detection method is needed for detecting a CE problem (but its accuracy is not perfect).

g. Each *dM* is independent of other detection methods in this experiment.

   We assume that each detection method is independent of other methods.

5.2 Experimental Settings

The definition of a CE table is given in Table 4. In this experiment, the following elements of the CE table are used:

a. The occurrence frequency (F_x) of a CE *x*
b. Damage (D_x) of a CE *x*
c. Detection cost (dC_{xy}) of a detection method *y* for CE *x*
d. Detection accuracy (dA_{xy}) of a detection method y for CE *x*

Based on the elements of the CE table and assumptions, we can implement Eq. 9 which is rewritten as Eq. 10 to determine viability-based detection policy $(dP)_T$.

$$\kappa = \frac{F_x \times D_x}{(1 - dA_{xy}) \times F_x \times D_x + dC_{xy}} \tag{10}$$

if $\kappa < 1$ Do not apply $dM_x^{''}$ to detect

if $\kappa \geq 1$ Apply $dM_x^{''}$ to detect ($dM_x^{''} \rightarrow dM_x^{'}$)

where κ is the applicability factor of *dM*

$dM_x^{''}$ is an applicable detection method that is selected from dM_x to detect CE *x*;

$dM_x^{'}$ is a detection method that is selected from dM_x to detect CE *x* and can maintain higher viability of a Co-U.

Table 5 shows an illustration of variable settings in this experiment. Here, because we only apply one *dM* to detect a CE, for each possible CE problem, we list only one applicable *dM* for it. If more than one *dM* can be used to detect a particular CE, we can select the *dM* with the highest *ß*. Table 5 is the same as the CE table stored in the CE knowledge base. Before performing the CE detection, the DPEM of a Co-U will retrieve this CE table, evaluate *dM*, and recommend a $(dP)_T$ based on Eq. 10. In this example, two CE problems are not detected (CE 1 and CE 10) and the $(dP)_T$ is generated as {*dM2, dM3, dM4, dM5, dM6, dM7, dM8, dM9*}.

Table 5: Illustration of the variable setting in the experiment

CE *x*	$\frac{1}{F_x}$ (sec.)	D_x ($)	*dM*		$(I)_{dM_x^{''}}$	$(II)_x$	Applicability factor κ	Detect or Not
			dC_x ($/sec.)	A_x (%)				
1	100	100	10.00	1.00	10.00	1.00	0.10	No
2	100	200	0.20	0.94	0.20	2.00	9.97	Yes
3	150	300	0.30	0.97	0.30	2.00	6.66	Yes
4	150	400	0.40	0.98	0.40	2.66	6.66	Yes
5	200	500	0.50	0.91	0.50	2.50	4.99	Yes
6	200	600	0.60	0.92	0.60	3.00	4.99	Yes
7	250	700	0.70	0.93	0.70	2.80	3.99	Yes
8	250	800	0.80	1.00	0.80	3.20	4.00	Yes
9	300	900	0.90	0.99	0.90	3.00	3.33	Yes
10	300	100	1.00	0.93	1.00	0.33	0.33	No

Detection-based dynamic viability (*DV*)

In order to analyze detection policy in terms of risk management, in this research, we use detection-based dynamic viability (*DV*) that is revised from dynamic viability *di(t)* (Eq. 2) to appraise the Co-U performance. The *DV* can be defined by the following equation.

$$DV(t) = W_i(t) - \sum_{x}^{N} dC_x^{''}(t) - \sum_{x}^{N} D_x(t) \tag{11}$$

where *DV(t)* is the detection-based dynamic viability in time *t*;

$W_i(t)$ is the accumulated reward in time *t*;

$dC_x^{''}(t)$ is the accumulated detection cost of an applied detection method regarding CE *x* in time *t*. It is equal to $dC_x \times t$;

D_x *(t)* is accumulated damage of CE *x* in time *t*;

N: the total number of CE problems that a Co-U tries to detect.

Based on Eq. 11, *Wi(t)* is the reward accumulated by performing correctly a task in time *t*. In this experiment, we calculate the *DV* by accumulating *Wi(t)* and subtracting two values. One is accumulated detection cost, $dC_x^{"}(t)$; another is the accumulated damage D_x *(t)* of a CE that has occurred but was not detected. This means that the dynamic viability should be affected by the accumulated detection costs and damages.

AutoMod simulation tool [AutoMod, 1999] has been used to simulate a single Co-U. A simulation program was developed to calculate the DV of a Co-U. The CE occurrence cycle times ($1/F_x$) of CE problems are assigned by Automod's exponential distribution. A simulated Co-U continually calculates DV until the end of a run. The experiment setting and variable setting are shown in Table 6 and 7.

Table 6: Experiment settings

Parameters	Value
The number of CE problems	10
Reward of performing a task	30 ($/sec.)
Runs of each combination	5 runs
Simulation time for each run	One hour (3600 seconds)

Table 7: Experiment variables of CE table

Experiment Variables			
Average Detection cost (*dC*) ($/Second)	Damage (*D*) ($)	CE cycle time ($1/F_x$) (Seconds)	Detection accuracy (dA_x) (%)
Assigned from 0.1 to 0.5, the interval is 0.5 (5 groups of *dC*)	Assigned from 10 to 100, the interval is 10 (10 groups of *D*)	Randomly assigned by exponential distribution ($1/F_x$ is assigned form 100 to 250 seconds randomly)	Randomly assigned by triangular distribution (0.94, 0.98, 1)
1. DV of a Co-U is calculated over 3600 seconds (one hour) 2. All combinations of various detection costs and damage values are simulated. The number of combination is 5X10=50 and each combination has 5 runs. 3. Totally, 250 simulations for each detection policy are run in this experiment.			

The purpose of this experiment is to determine that the viability-based detection policy ($dP)_T$ can maintain higher dynamic viability based on the decision-making logic of DPEM. Two simulations are run based on two detection policies. They are Detection Policy #1 - detecting all CE; Detection Policy #2 – viability-based detection policy ($dP)_T$; i.e. following Eq. 10. All parameter settings and variable settings are the same in the two simulations except for the value level of average detection cost and damage. The experiment results and ANOVA results based on these two detection policies are addressed in the following sections.

5.3 Experiment Results

In this experiment, there are 5 value levels of "average detection cost", 10 value levels of "damage of a CE", and 2 kinds of detection policies. Therefore, there are 100 combinations and totally, 500 simulations runs (each combination has 5 runs). Partial experiment results are shown in Table 8 (full listing is available in Yang, 2004), and corresponding ANOVA results for the complete results are shown in Table 9.

Table 8: Experiment results (partial)

Detection Policy *dP*	Damage from CE *(D)($)*	Average Detection Cost *(dC)($)*	Detection-based dynamic viability (*DV*)($)
1	10	0.1	67779.94
1	10	0.1	69481.13
1	10	0.1	68607.90
1	10	0.1	68190.35
1	20	0.1	68166.87
1	20	0.1	68356.47
1	20	0.1	67399.98
1	20	0.1	67466.63
1	30	0.1	67200.41
1	30	0.1	67391.90
1	30	0.1	68758.92
1	30	0.1	69179.91
…..	….	….	….
2	100	0.5	56188.29

Table 9: 3-way ANOVA output for experiment results

Class	Levels	Values
Detection Policy	2	1, 2
Damage	10	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Average Detection Cost	9	0.1, 0.2, 0.3, 0.4, 0.5
One Variable: Detection-based dynamic viability (*DV*) Number of observations 500		

The ANOVA Procedure Dependent Variable: Detection-based dynamic viability (*DV*)					
Source	DF	Squares	Mean Square	F Value	Pr > F
Model	63	11530078927	183017126	43.05	<.0001
Error	436	1853489125	4251122		
Corrected Total	499	13383568052			

R-Square	Coeff. Var.	Root MSE	Detection-based dynamic viability (*DV*) Mean
0.861510	3.313937	2061.825	62216.77

Source	DF	ANOVA SS	Mean Square	F Value	Pr > F
Policy (*dP*)	1	1295119460	1295119460	304.65	<.0001
Damage (*D*)	9	582623798	64735978	15.23	<.0001
Cost (*dC*)	4	8280286927	2070071732	486.95	<.0001
Policy*Damage (*dP*D*)	9	204392876	22710320	5.34	<.0001
Damage*Cost (*D*dC*)	36	270534925	7514859	1.77	<.0001
Policy*Cost (*dP*dC*)	4	897120941	224280235	52.76	<.0001

The 3-way ANOVA results (

Table 9) indicate that all null hypotheses of each factor; i.e., detection policy (*dP*), damage (*D*), and average detection cost (*dC*) are rejected. This means that *dP*, *D*, and *dC* are significant and affect detection-based dynamic viability (*DV*). Next, we compare detection policies based on average detection cost and damage analysis.

5.4 Comparison of Detection Policies

The ANOVA results in

Table 9 already indicate that both *dC* and *dP* are significant in affecting the *DV* in this experiment. Figure 4 illustrates the comparison of detection policies (*dP*) based on different average detection costs (*dC*). The results show that both *DV* of dP1 and of dP2 decrease as *dC* increases, on average. It is intuitive, since a higher *dC* means a Co-U has to spend more on detection. In addition, in comparing dP1 and dP2, we find that *dP2* maintains relatively higher *DV* as *dC* increases, and is always better than *dP1* in terms of *DV* from about *dC* > 0.14. It makes sense because when *dC* increases significantly, a Co-U is not likely to afford to detect all the CE problems, based on the selective dP2's decision logic (κ decreases).

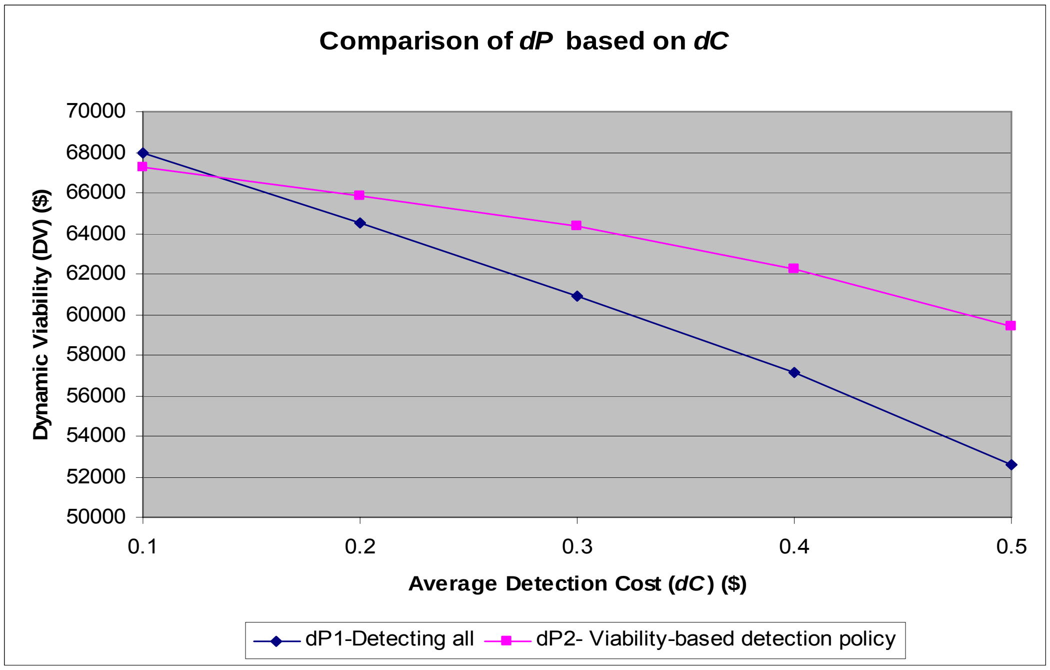


Figure 4: Comparison of *dP* based on different *dC*

Similarly, the ANOVA results in

Table 9 also indicate that both *D* and *dP* significantly affect the *DV* in this experiment. Figure 5 illustrates the comparison of detection policies (*dP*) based on different damage levels (*D*). The results show that both *DV* of dP1 and of dP2 decrease as *D* increases, on average. It is intuitive, since a higher *D* means a Co-U's operations will be harmed by higher damages if they are not detected and prevented. Besides, while comparing *dP1* and *dP2*, we can notice that *dP2* can maintain a relatively higher *DV* as long as *D* increases, because when *D* increases, a Co-U is more likely to attempt to detect the CE problems based on dP2's selective decision logic (κ increases). It also implies that policies of detecting-CE are more cost-effective than not-detecting-CE when the *D* increases.

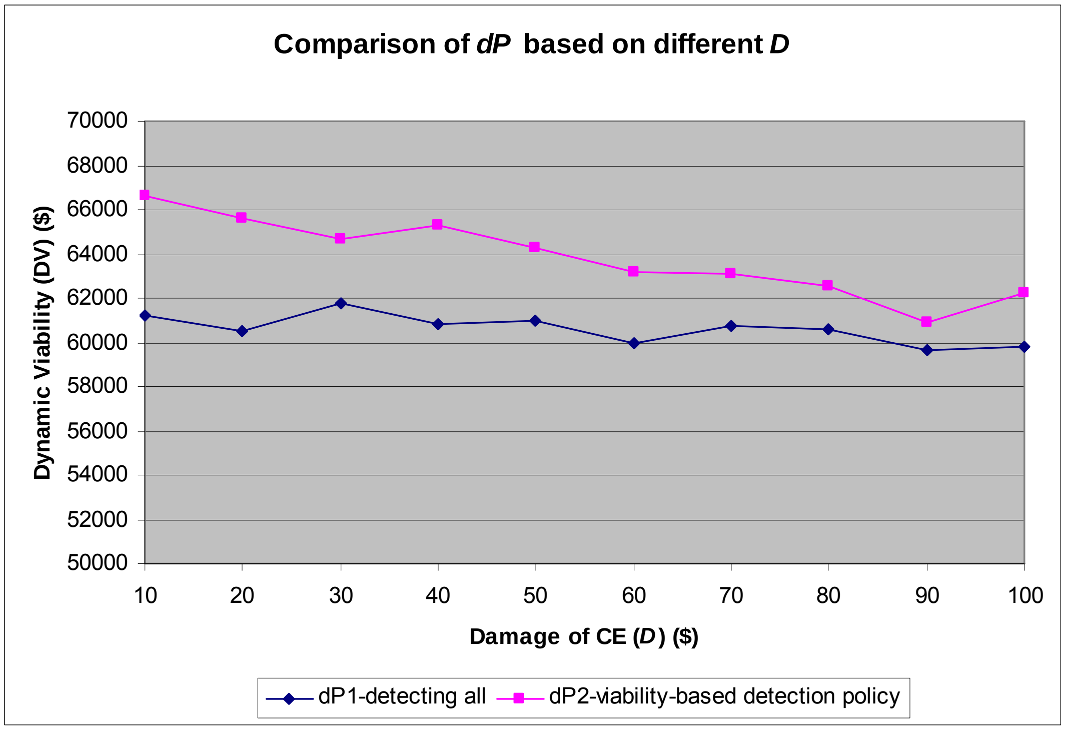
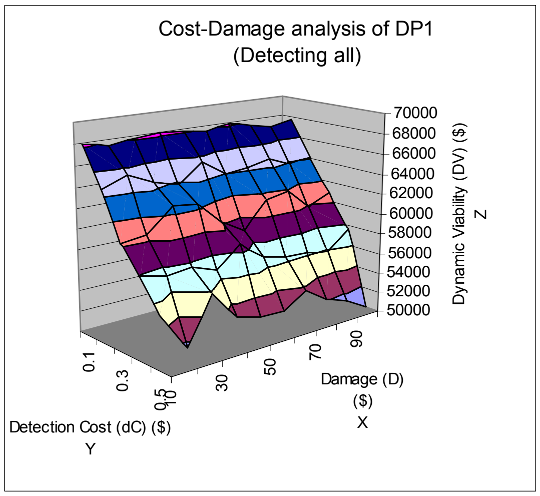


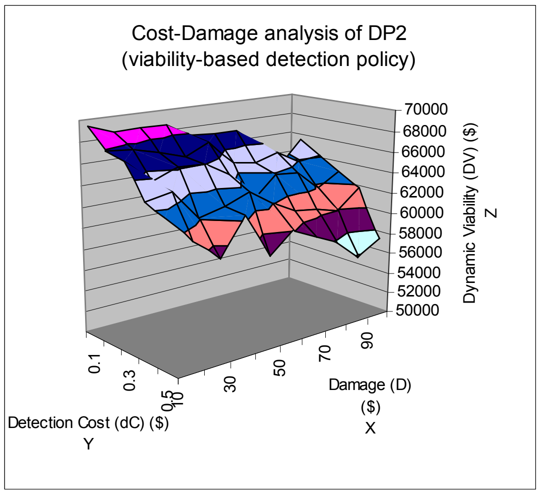
Figure 5: Comparison of *dP* based on different values of *D*

We can now plot *DV* values on a 3D chart both relative to *dC* and *D* (Figure 6). The *X* and *Y* axes in Figure 6 (a) and (b) represent D and dC, respectively. The *Z* axis of Figure 6 (a) and (b) shows the level of *DV* values. Again, in Figure 6, *dP2* can maintain relatively higher *DV* since the *DV* surface of *dP1* is steeper than the surface of *dP2* overall. The largest difference between the *DV* of *dP1* and *dP2* is found when *D* is relatively low (*D* = 10) and *dC* is relatively higher (*dC* = 0.5). It is reasonable, because in the reverse (very high damages, low detection cost), it makes more sense to detect all CE problems.

In order to maintain relatively higher levels of *DV* and cost-effectiveness of CE detection, a new, selective detection policy has been developed and studied. According to the above discussion, a selective detection policy made by DPEM can maintain a higher *DV* especially when *D* is relatively lower and *dC* is relatively higher. This conclusion can be proven by using a linear programming procedure to solve this optimization problem for maximum viability. Although this problem can be modeled as deterministic and linear programming technique can be applied to solve this problem, based on DPEM's decision logic, a relatively simple decision-making process for the actual, non-deterministic behavior can be applied by evaluating κ of each *dM*. Furthermore, in a practical situation, the timing of deciding which CE should be detected might vary. The evaluation method and measures developed in this research can be applied in this type of dynamic situations instead of deterministic optimization calculation. Especially, when the number of CE problems is relatively large, the viability-based selective detection policy can enhance the efficiency of the detection decision process when a Co-U needs to decide which detection policy, *dP,* it should prefer for the given case.



(a)



(b)

Figure 6: Comparison of *dP1 and dP2* based on different *dC* and *D*

In this section, the results of the experiment show that DPEM can facilitate the ability of a Co-U to decide how to determine a recommended detection policy for CE detection. The applicability factor (κ) of each detection method dM_x can be used to decide whether this dM_x should be applied or not. By applying selectively recommended detection policies, the impact on individual Co-Us and the total impact on the Co-net of the company, in terms of beneficial viability which represents profitability, can be measured and optimized.

6. Conclusions and Recommendations

In this research, we focus on the analysis of conflict and error problem in a task planning collaborative environment, and their impact on company competitiveness (which is measured here as dynamic viability). First, we define conflict and error problems in task planning collaboration. In order to analyze a CE problem after task allocation, taxonomy of the CE problems is developed. Based on the taxonomy, different types of conflict and error problems in task planning can be categorized. Relative cost, damage and detection method analysis also show that applying appropriate, rationally selective detection methods to detect CE problems is beneficial.

In addition to analyzing and categorizing CE problems, we identify the CE table for storing the CE information. When a Co-U attempts to detect CE problems, related information and experiences can be retrieved from the CE tables to enhance the detection performance. This table can become a knowledge base for detection. In addition, detect-ability (*ß*) measure is introduced in this research. Detect-ability (*ß*) can be used to evaluate the competency of detection methods. Another measure, dynamic viability (*DV*), can be used to evaluate the CE detection performance of the Co-U.

Maintaining a higher level of *DV* when performing CE detection means that the overall cost-effectiveness of CE detection is higher.

The DPEM can evaluate particular CE detection methods and recommend a better detection policy, viability-based detection policy, to decide how to apply alternative, applicable detection methods. The important objective of a DPEM is to maintain higher viability when it performs CE detections. By considering the preventable damage of a CE and its detection cost, the Co-U can have greater flexibility in deciding on its detection policy.

The experiment was designed to simulate the decision-making process when a Co-U selects a detection policy before actually detecting CE problems. It focuses on the performance comparison between a common detection policy and the viability-based detection policy proposed in the Section 4. The common detection policy is to try and detect all possible CE events, regardless of the relative comparison of the average detection cost of applied detection methods and relative to the magnitude of potential damages resulting from un-detected and un-prevented CE events. On the other hand, viability-based detection policy $(dP)_T$ considers the possible damage of CE and average detection cost in order to maintain higher dynamic viability of a Co-U, even when certain CE cases are not going to be detected. In this experiment, for both detection policies, a DPEM retrieves CE tables to obtain CE information for a particular task before performing a detection process. After the Co-U evaluates alternative detection methods, it determines a viability-based detection policy $(dP)_T$ that decides how to apply detection methods on CE detection *and* maintain the higher viability of a Co-U.

Experimental results indicate that “detecting all CE problems” even with the recommended detection method, is not always useful if the detection costs are relatively high or the damages from the CE events are relatively low. By implanting the DPEM decision-making logic, a Co-U can improve the detection policy to maintain overall higher viability, which represent an overall positive impact on the company.

Future Research

Conflict/error detection is an initial stage of conflict/error management [Balakrishnan et al., 1994]. Solving or recovering the detected CE problem in the collaboration environment is a following mission. Several undeveloped areas which need to be investigated are:

a. A well-organized conflict/error knowledge base is necessary to improve the conflict/error detection and resolution process. Knowledge acquiring and query mechanisms that enable collaborative members to deposit and retrieve detection information are important issues.

b. Since the cost of resolving CE problems might be also costly and significantly affect the profitability of a company, including it in a proposed Detection Policy Evaluation Mechanism will be the next step to improve this mechanism.

c. Evaluation and comparison of the proposed measures with other quality control measures should be investigated in the future work.

References

AutoMod User's Manual volume 1 and 2, 1999, AUTOSIMULATIONS, Inc.

Anussornnitisarn, P., 2003, Design of Active Middleware Protocols for Coordination of Distributed Resources, Dissertation for Ph.D., Purdue University, West Lafayette, IN., U.S.A.

Anussornnitisarn, P., and Nof, S. Y., 2003, e-Work: The Challenge of the Next Generation ERP Systems, *Production Planning and Control*, v. 14, n 8, pp. 753-765.

Bakken, D. E., 2003, Middleware, Chapter in *Encyclopedia of Distributed Computing, Urban J. and Dasgupta P.(editors)*, Kluwer Academic Publishers.

Chen, X., and Nof, S.Y., 2004, Design of Active Middleware for Error and Conflict Detection, Research Report GM-CEDM-Report 04-1, March.

Dellarocas, C., and Klein, M., 2000, An Experimental Evaluation of Domain-Independent Fault Handling Services in Open Multi-Agent Systems, 4th Int. Conf. on Multi-Agent Systems.

Huang, C. Y., and Nof, S. Y., 2000, Autonomy and Viability-measures for Agent-based Manufacturing Systems, *International Journal of Production Research*, 38(17), 4129-4148.

Klein, M., 2000, Towards a Systematic Repository of Knowledge About Managing Collaborative Design Conflicts, *Proc. of the 6th Int. Conf. on AI in Design*, Worcester, MA, June 26-29.

Klein, M., and Dellarocas, C., 2000, A Knowledge-based Approach to Handling Exceptions in Workflow Systems, *Computer Supported Cooperative Work*, v 9, pp. 399-412, Kluwer Academic Publishers.

Luo, Z., Sheth, A., Kochut, K., and Arpinar, B., 2000 ,Exception Handling in Workflow Systems, *Applied Intelligence*, v 13, n 2, pp. 125-147, Sept.-Oct. 2000.

Luo, Z., Sheth, A., Kochut, K., and Arpinar B., 2003 ,Exception Handling for Conflict Resolution in Cross-Organizational Workflows, *Distributed and Parallel databases*, v 13, n 2, pp. 271-206, May 2003.

Nof, S. Y., 2003, Design of Effective e-Work: Review of Models, Tools, and Emerging Challenges, *Production Planning and Control*, 14(8), 681-704.

Ostergaard, K. and Summers J. D., 2003, A Taxonomic classification of collaborative Design Process, International Conference on Engineering Design, ICED 03, Stockholm.

Robin, V., Girard, P., and Barandiaran, D., 2004, Performance Evaluation of the Collaborative Design Process, *11th IFAC Symp. INCOM'04, Salvador, Brazil, April.*

Smith, R. G., 1980, The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Transactions on Computers*, C-30(5), 372-385.

Yang, C.-L., Conflict and Error Detection Protocol with Active Middleware, M.S.I.E. Thesis, Purdue University, August 2004.