CONCURRENT ENGINEERING: Research and Applications

A Constraint-driven Execution Plan for Maximizing Concurrency in Product Development

Jinmin Hu,[1,*] Jianxun Liu[1] and Brian Prasad[2]

[1]*CIT Lab, CS Department, Shanghai Jiao Tong University No. 1954, Huashan RD, Shanghai, 200030, China*
[2]*Spec2Market Solutions, P.O. Box 3882, Tustin, CA 92782, USA*

Abstract: One of the major goals of Concurrent Product Development (CPD) is to shorten the design cycle time by increasing more task overlaps of the ensuing design activities. Increasing task or activity overlap often hinges on two timing questions (a) ''When does the downstream activity require certain information from its upstream activities and (b) when will the required information on an activity be available for scheduling the next dependent task ?'' The paper discusses a constraint-driven execution plan to handle the concurrency during scheduling of various dependent tasks. This constraint-driven execution plan is designed to relieve the alignments of start and end points commonly encountered during scheduling of the concurrent tasks. It is not essential that an activity will start only when a dependent activity is fully finished. In constraint-driven execution plan, a task or an activity can be initiated as soon as the required information is available rather than after the completion of all the upstream activities dependent on it. The paper shows that if we follow this execution plan during a CPD process, concurrencies among the ensuing design activities can be maximized. The paper also discusses how this execution plan can be used in a context of a workflow management system (WfMS). Employing this plan, it is possible to easily transform most of the event-condition-action (ECA) rules-based workflow model – from its activity completion event-driven forms – into an information-constraint-satisfaction-driven forms. This execution plan, thus, enables an event-driven WfMs to support concurrency of tasks during product development and concurrent tasks' scheduling.

Key Words: Concurrent Engineering, Concurrent Product Development, Information Constraint, Workflow Management System, ECA Rules.

1. Introduction

In order for companies to win the time-based competition, most enterprises need to introduce new competitive product quickly and consistently, to the global marketplace. New product development is becoming an essential requirement to maintain a high degree of competitiveness. In order to support such goals, many companies have used and are using Concurrent Engineering (CE) techniques to support fast, less costly, and improved quality in product development. The most important goal of concurrent product development (CPD) is to shorten product development cycle time. To reach this goal, there are several different approaches to implementing CE and reducing time during CPD. Very often, some kinds of information management systems are generally built around CAD systems to manage the product content and design model information [3]. In addition, some information systems known as PDM systems are often employed to provide information sharing and to facilitate cooperative design. To reduce or eliminate engineering changes and redesigns, DFX (Design For Manufacturing, Assembly and etc.) methods and analysis tools are generally used to account for the manufacturing and assembly process in the early product design and development phase [9]. However, to shorten the entire time-to-market in product development, the life-cycle process needs to be carefully planned and managed.

A design process of a complex product is very complicated due to the presence of various types of constraints. The constraints can be present as restrictions on time, as relations on product geometric structure, or in the form of design or manufacturing specifications. Depending on the impacts they inflict, they are called as information constraints, resources constraints, and so on. Information constraints are imposed on the design activities to identify start and end timing points so as to align them concurrently. Because of their inherent time dependencies, it is often difficult to independently carry out some activities without the help of others. The idea of information constraints is to make the start and end points of each activity and their time dependencies visible. Thus, during job scheduling, when the information constraints are specified on associated tasks, such activities are carried out in an appropriate order maintaining the needs for information transfer among the associated tasks at appropriate time points. Chaining of activities in this manner is called herein as ''execution plan.''

*Author to whom correspondence should be addressed.

1063-293X/03/04 0301–12 $10.00/0 DOI: 10.1177/1063293X03038568

There are many ways to create such execution plans. Some execution plans are better than others. In CE, it is important that tasks are scheduled concurrently. One way to achieve these objectives is by forcing all the constraints imposed on an activity to be fulfilled prior to the start of this activity. Will this be a good plan ? The authors have found that a better plan can be devised by permitting information constraint insertions to take place at intermediate time points, other than the start and end time points of an activity.

This paper mainly discusses the information constraint in product development life-cycle activities. Within a product development (say a life-cycle) process, usually, downstream activities require information; say for example, some design parameters form a set of upstream activities. Typically, during a life-cycle process, the downstream activity cannot be started before all the upstream activities are completely finished. But the fact is that this type of information constraint is often too limiting in meeting the overall project goal. The idea proposed in this paper is to initiate an activity as soon as possible even when only partial constraint on an activity is satisfied. The essence of this idea stems from the fact that more time overlaps of the design activities reduce the overall completion time. In reality, every design activity within a life-cycle process need not start only when a prior activity is finished. The constraints imposed on these activities only at "start and end points" are often too restrictive from timing aspect. The authors have found that including intermediate time points in addition to start and end points of an activity, while applying information constraints, is the major force to increase the degree of concurrencies. When time restrictions on the constraints are relaxed from start and end points to include additional intermediate timing points, it does significantly alter the resulting execution plan. One then considers the dependencies of a set not with respect to its terminal points (at the beginning or start of an activity) but with respect to what would be an optimal timing to launch a next set of activities when the first set is not even fully completed. However, in considering such options, the degree of concurrency in the execution plan is significantly increased and the time of completion is accordingly reduced. By considering more intermediate time points and strategically scheduling activities by satisfying information constraints at those points, it can result into a better execution plan. Better in the sense that as soon as the associated information constraints are satisfied on each of the chained activities in the execution set, it is possible to maximize the degree of concurrency and significantly reduce the clock time for CPD.

There has been a lot of research on product development process (re-) engineering and product development project management. However, little research attention has been focused on the process of managing execution of an activity set. Project enactment and managing activity execution is very important to the success of any CE implementation. Workflow technology has already been proven to implement successfully business process management or other process-oriented project management [1,2,14]. Many researchers used workflow management system (WfMS) to support development project management as well [10,13]. In this paper, authors have used a workflow management system to help develop a better plan for concurrency and manage the execution of the ensuing design activities. The resulting workflow execution is driven by the satisfactions of the information constraints on the associated activity set.

The rest of the paper proceeds as follows. Section 2 briefly introduces some related work, especially, some activity dependency analysis approaches. Section 3 presents the concept of task overlapping and information constraint within concurrent product design context. The information constraint-driven workflow execution semantics is proposed and compared to the common workflow execution semantics by using measure of concurrency (MOC). Section 4 extends the ECA-based workflow model to support the constraint-driven semantics. Attempt is also made to extend the existing workflow system architecture by adding constraint management and by creating modules to support the extended ECA rules. The last section of this paper deals with discussion and conclusion.

2. Related Work

There are several approaches to modeling life-cycle processes and analyzing activity dependencies in product development and project management. The Program Evaluation Review Technique (PERT) and the Critical Path Method (CPM) [15] are some popular approaches to modeling life-cycle process and managing integrated product development (IPD) project. Steward [16,17] developed a design structure matrix (DSM) as a tool to represent and analyze activity dependencies of a design project. The DSM is a binary square matrix with m rows and columns, and n non-zero elements, where m is the number of design activities and n is the number of information flows (or dependencies) going from one design activity to the other. If there exists an information dependency from activity i to j, then the value of element ij (column i, row j) is unity (or marked with an X). Otherwise, the value of the element is zero (or left empty). Eppinger [4] used DSM concept to describe the information flows of a project rather than concurrent scheduling its workflows. It did not address the question: "What information does one need from other activities before one can complete the current

activity at hand ?" Since then, many authors [18,21] have explained how DSM concept works and how to use it to make life-cycle processes more efficient. However, these works did not consider partial overlap as a solution to exploit activity dependencies while scheduling workflow. Partial overlap forces one to consider timings of information release and insertion points during a life-cycle process for a product development.

Krishnan [11] presented a model-based framework to manage the overlapping of coupled product development activities. It addressed two questions: (a) how early an upstream information set could be finalized and (b) under what conditions could preliminary information of upstream action be useful for downstream action. But, their overlapping model did not extend to multiple activities or concurrencies. In fact, their theory of overlapping activities is a complement to this work, as it can be used to estimate when the upstream activities can provide the information to a downstream activity.

Today, WfMS is very popular to support information or data flow during project management [8,10,13]. Nevertheless, increasing concurrencies of tasks/activities during a workflow execution is still a research problem. Most existing workflow execution plans, reported so far, is based on the "control flow" but not on timing of "data flow across various activities" or their subsets. In other words, they are event-driven but not information-constrained or time-driven. A concurrent-constraint-satisfaction process during a workflow is studied in this paper. This paper will show how to extend the current workflow model to include concurrent data flow, how to satisfy information constraints at intermediate points along a path of an activity set, and how to implement a constraint-driven execution plan semantics.

3. Information Constraint and Execution Plan Semantics

3.1 Concepts of Activity Dependency Based on Information

Concurrent engineering is a process where information from different domains (related to a product development life cycle, including design and manufacturing) needs to be passed along onto each other and to the associated workgroups that manage the process. Such types of information, where one activity is dependent on other, are characterized herein as sets of "information constraints." Similar definitions have been used in the past. For instance, topological attributes of a specific form feature were represented as constraints from process planning point of view; similarly a product-targeted cost is another constraint [6] used in product and portfolio planning. Besides satisfying dependencies among its constituent activities, such constraints also provide a foundation for requirement specification in a product development process. Constraints also state the dependencies on activities and associated execution plan in a workflow.

The constraints, discussed in this paper, are the information constraints imposed for the concurrent execution of the design activities in a workflow. Information constraint is one of the major constraints that directly impact how a series of activity will be executed, which in turn determines what degree of concurrency one can achieve using this plan. Another common constraint used in a workflow resource constraint. In this paper, we will be discussing information constraints mainly. Information constraint is defined here as a statement of time dependencies among activities of a CPD process. Time dependencies can be generated from any type of constraints, but information constraints provide a level of dependency, which is not generally found in a typical design process except those based on concurrent product development.

Definition 1 (*require information*). If an execution plan of an activity "A" needs information x, *then a* binary relation can be written as *require* (A, x). *It means that* activity A requires information x.

Definition 2 (*release information*). If an execution of an activity "A" releases information x, then a binary relation can be written as *release* (A, x). *It means that* information x is released by activity A.

Definition 3 (*activity dependency based on information*). If an activity A_2 requires information x, *which in turn*, is released from Activity A_1, in other words, if there is an information dependency between activity A_1 and A_2, then a relation can be written as *depends_on_for* (A_2, A_1, x). It means the execution of A_2 depends on the execution of A_1 because of information constraint x. A_1 releases x and A_2 requires it.

Proposition 1 *Require* (A_2, x) $\wedge$ *Release* (A_1, x) $\rightarrow$ *depends_on_for* (A_2, A_1, x).

3.2 Modes of Concurrency

Product development process is a process of gradually building up the right information and linking up the process activities with required personnel skills so that the project can be finished in time. However, it takes time to gradually build-up the information in a form that can be readily used by any skilled team member. Overlapping staggers these resources so that they are ready when information is complete and available in the form to be used. It would be a waste of resources if skills are available but the information is incomplete. There are three distinct possibilities in which a build-up and a transfer of information can take place. Prasad [20]

has shown the following three forms of overlapping in CPD.

- ***No Overlap***: There is no overlap; information is gradually ramp-up for an activity until it is complete, then it is transferred to the next activity where it is used to build up a new grade of information to support subsequent activities. This is called series transfer and the resulting process is called sequential process. It is suitable when the two activities are dependent, that is, information from one activity is required to support the second activity.
- ***Partial Overlap***: The transfer of information takes place during a build-up of an activity, when information ramp-up is only partially complete. A series of interactive transfer continues to take place until the information ramp-up is fully complete. At that point the information transfer from one activity ceases but the information build-up for the second activity continues. This is suitable for activities that are semidependent. The degree of transfer depends upon the degree of independence. If the two activities are completely independent, there is no transfer. If not, coupling is preserved and there may be a series of transfers. The resulting process is called "parallel process."
- ***Complete* (100%) *Overlap***: If the coupling between upstream and downstream activities can be removed, the two activities can start at once and run simultaneously. A series of interactive transfer may take place while the information is being ramped-up on each side. This is suitable when the two activities are independent. This type of parallel overlap is called mutual and the resulting process is called simultaneous process. A supplementary overlap is when a series of transfers between two parallel activities takes place to supplement the start (an information ramp-up) of a third activity.

The suitability of one or the other forms of overlaps depends upon:

(a) Whether or not there is a dependency between the activities
(b) How can one activity provide enough information for a second activity to get started early;
(c) Whether or not pertinent skills are freed-up or available, and
(d) Whether or not an activity is on a critical path. Without the overlap more time is needed since:
    (i) Supplied information may not be in the right format
    (ii) May not have the right content (as in the case of an information ramp-up) and,
    (iii) More time is needed to digest the information supplied. With some overlap, the time to completion can be significantly reduced.

3.3 Measure of Concurrency (MOC)

Before we analyze the concurrency of different execution semantics, let us define some concepts and equations to use for concurrency analysis. Prasad [12] defined a MOC and a related theory for CPD. He proposed seven principles to increase concurrency for product design based on his MOC theory. In this paper, we have adapted some of his MOC concepts and related equations – as reported in the AIEDAM paper [12].

Given an activity set:

$$A\text{-}set \equiv [a_1, a_2, a_3, \ldots, a_i, a_j, a_{j+1}, \ldots, a_n], \tag{1}$$

where a_i is the ith activity.
Let us also denote:

$$\begin{aligned} &ts_i \text{ as the start time, the time when an } i\text{th activity,} \\ &\quad a_i, \text{ starts;} \\ &te_i \text{ as the end time, the time when an } i\text{th activity,} \\ &\quad a_i, \text{ ends.} \end{aligned} \tag{2}$$

Then, duration of an ith activity, also called the lead-time, di, can be expressed as:

$$d_i = te_i - ts_i \tag{3}$$

We can arrange the A-set as a precedent A-set, which satisfies:

$$\begin{aligned} &\text{For any } a_i \text{ and } a_j\text{: If } ts_i < ts_j, \quad \text{then } i < j; \\ &\text{And, if } ts_i = ts_j \text{ and } te_i = te_j, \quad \text{then } i < j \end{aligned} \tag{4}$$

Now, we denote c_i as the "MOC" between any two consecutive activities, a_i and a_{i+1} the MOC or overlap can be expressed as follows:

$$c_i = 1 - (ts_i - ts_{i-1})/d_{i-1}, \tag{5}$$

where d_{i-1} is the duration of an activity a_{i-1}. Using Equation (3), d_{i-1} can be expressed as

$$d_{i-1} = (te_{i-1} - ts_{i-1}) \tag{6}$$

If T_i is the clock time of an ith activity. The clock time is the time an ith activity, a_i, takes from start ($t = 0$) to its finish. Following this definition, then $T_1, T_2, T_3, \ldots, T_n$, can be expressed as

$$\begin{aligned} T_1 &= d_1, \\ T_2 &= \{d_2 + d_1 \times (1 - c_2)\}, \\ T_3 &= \{d_3 + d_1 \times (1 - c_2) + d_2 \times (1 - c_3)\}, \\ T_k &= \{d_k + d_1 \times (1 - c_2) + d_2 \times (1 - c_3) \\ &\quad + \cdots + d_{k-1} \times (1 - c_k)\} \end{aligned} \tag{7}$$

$$T_n = d_n + \sum_{i=1}^{n-1} d_i \times (1 - c_{i+1}) \quad (8)$$

The Equations (7) and (8) provide a basis for computing the total product development time, T_k. In the following sections, we will use the aforementioned equations to analyze the concurrencies and compute the lead time of the whole life-cycle process in different execution semantics.

3.4 Common Execution Plan Semantics

There are many commonly-used semantics of a constraint-driven execution plan. Figure 1 shows an example of a typical product development process. Irrespective of a particular semantics employed for its representation, it contains seven design activities. The bubbles S and E are dummy activities, which seperately represent the start and the end points of the process.

Following the common execution semantics of DAG: (Direct A-Cyclic Graph) based process model, the arrowed edges and rectangular blocks in Figure 1 are drawn to represent an execution plan. The "execution plan" direction is used herein to indicate that only after the set of upstream activities is finished, the downstream activity set can be started. Also shown in the figure, are the variables x_i or y_i placed on the arrowed edges. The variables represent the information dependencies between these activities. The x_i or y_i marked on the edge is the information that is released by the upstream activity(s) and is required by the downstream activity(s). In the scope of the workflow technology, these information are called workflow relevant data [23]. Here, besides the workflow relevant data used by workflow engine, the dependency information among the design activities are also included. The dependency in this case is often represented by application data required by some activity(s) and generated by other activities. Such design results are commonly stored in a PDM system.

According to our definitions in Section 2.1, the dependency of the DAG-based process model, shown in Figure 1, can be transformed into the following seven dependency relations:

depends_on_for (A_4, A_1, x_1);
depends_on_for (A_4, A_3, x_3);
depends_on_for (A_5, A_2, x_2);
depends_on_for (A_5, A_4, x_4);
depends_on_for (A_6, A_4, y_4);
depends_on_for (A_7, A_5, x_5);
depends_on_for (A_7, A_6, x_6);

The durations of each activity shown in Figure 1 are given in Table 1. The activity durations are usually set by the project managers (through estimating) during planning of the design process. A list of activity durations for this example is separately given in Table 1. They are also shown in Figure 2 pictorially – to be used in discussing concurrency and in computing total time for completion.

The common execution semantics of DAG-based process model is that "activity B cannot be started until activity A is completed if there exists a dependency of B on A." Such a dependency relation is expressed as *depends_on_for* (B, A, x). Based on this semantics and the process graph shown in Figure 1, the following conclusions can be drawn:

(a) A_1, A_2, and A_3, can be initially started and executed simulaneously ($ts_i = 0$; $i = 1,2,3$);
(b) A_4 cannot be started until A_1 is finished;
(c) A_5 cannot be executed until A_2 is finished;
(d) A_6 cannot be started until A_4 is finished.
(e) A_7 can be started only after both A_5 and A_6 are completed.

By following this execution semantics, an activity execution plan of the process is shown in Figure 3 as Execution Plan (1).

The activity duration of A_i, d_i, the MOCs and T_i for execution plan (1) are shown in Table 2.

From Table 1, we can see that only A_1, A_2, A_3 are executed concurrently and the whole cycle for this process is $T_7 = 28$ (tus, time units). However, if we accurately analyze the information dependencies among these activities – not based on the assumptions that the information required and released only occurs

Table 1. Activity's lead time.

A_i	A_1	A_2	A_3	A_4	A_5	A_6	A_7
d_i	2	4	8	7	9	2	4

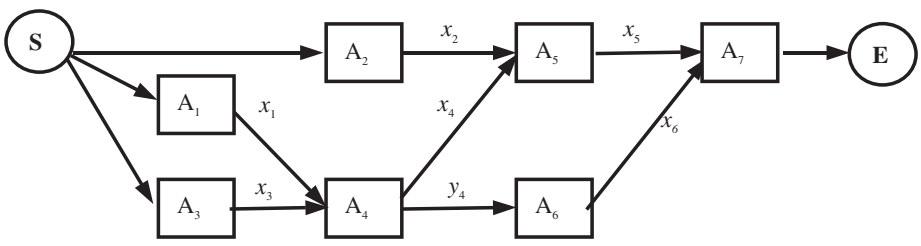


Figure 1. A 7-activity process diagram – showing activity dependencies and constraint variables.

at the begining of an activity or its end – we will find that this execution cycle can be shortened dramatically. The following section discusses how to make more partial overlaps possible.

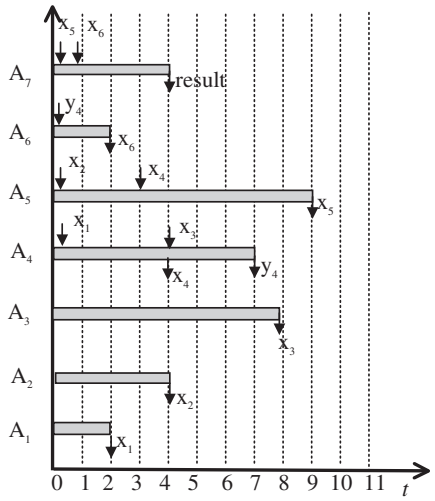


Figure 2. Activity lead-time and the individual relative time points of information required and released.

3.5 Information Constraint-driven Execution Semantics

To shorten the design cycle, authors have focused here on another question about design information, which is different from Eppinger's [4]. The two questions are: "When does the activity actually require the information generated by its upstream activities" and "when will the required information be available for the activity to use ?" To answer both questions, one should analyze the information dependencies not just based on two terminal points, which occurs at the beginning of an activity or at the end but based on other insertion points, which may occur at intermediate time points. Some design activities cannot be partially overlapped because we made an assumption that when a latter activity starts the former must end. Information constraints imposed on such activities due to dependencies are too strict from timing perspective. For example, when design activity *B* requires information *x* generated by activity *A*, one always thinks that activity *B* cannot be started before activity *A* is finished – by following the common execution semantics. However, to accurately analyze such dependencies, and impose a better set of

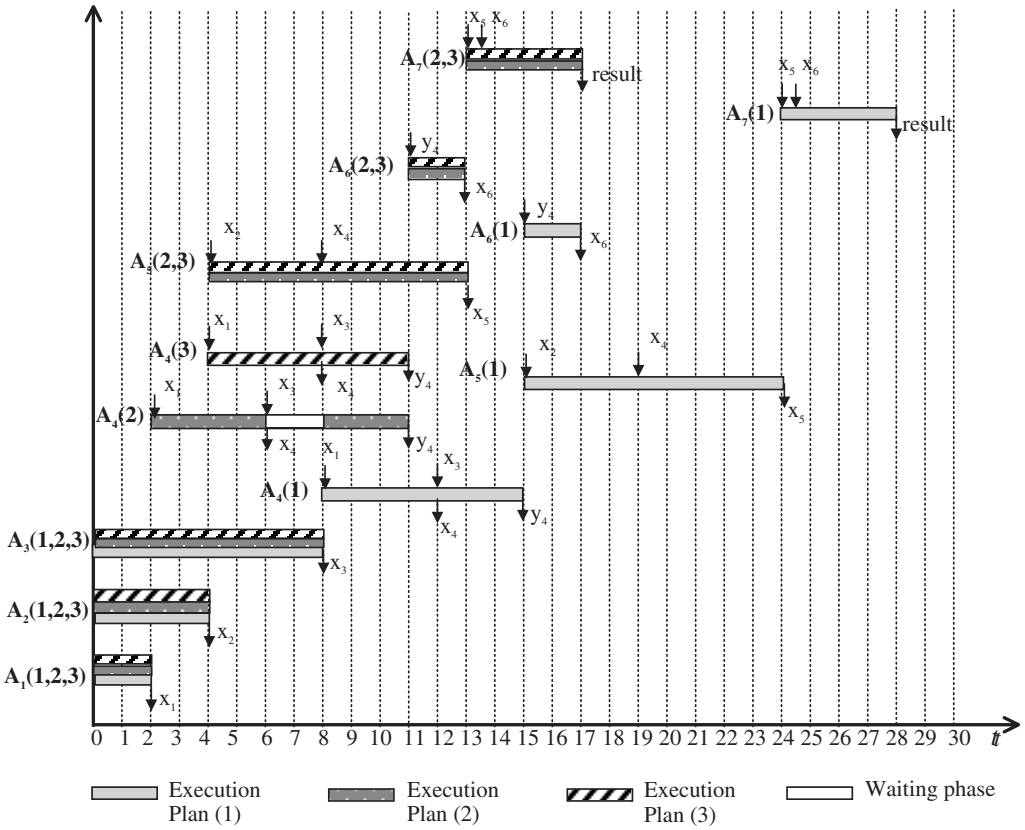


Figure 3. Activity execution following two different execution semantics. Execution Plan (1) –Traditional execution semantics; Execution Plan (2) – Constraint-driven semantics. Execution Plan (3) – Waiting phase in Execution plan (2) eliminated.

Table 2. The d_i, MOCs, and T_i for execution plan (1).

a_i	$a_1(A_1)$	$A_2(A_2)$	$a_3(A_3)$	$a_4(A_4)$	$a_5(A_6)$	$a_6(A_5)$	$a_7(A_7)$
d_i	2	4	8	7	2	9	4
ts_i	0	0	0	8	15	15	24
$c_i = 1 - (ts_i - ts_{i-1})/d_{i-1}$	–	1	1	0	0	1	0
T_i	2	4	8	15	17	24	28

a_i is ith activity in the arranged A-set which satisfies Equation (4). A_i denotes the activity A_i shown in Figure 1. In the rearranged A-set, A_6 is in the 5th position and A_5 is in the 6th position.

information constraints on overlapping activities, the following questions must be asked:

- When does activity A release information x?
- When does activity B require this information x?

If x is released by activity A not at the end (when activity A is finished) and if activity B requires x not at the start of the process when Activity B is launched, can we overlap part of the execution of these two activities? Hence, through accurate analysis of timing points in an execution plan, better concurrencies of activities can be attained.

Figure 2 specifies the relative estimated time points when the information required and released. The arrow with the variable $x_i(y_i)$ above the activity execution bar A_i represents the information required by A_i; the arrow with the variable $x_i(y_i)$ below the activity execution bar A_i represents the information released by A_i. The time points given are related to the time point of the start of individual activity. For example, as shown in Figure 2, information x_1 is required by A_4 as soon as A_4 is started to execute; x_3 is also required by A_4 but at the time point (4 tus, time units) after it is started; A_4 generates x_4 at the time point (4 tus) after it was started and y_4 in the end of the execution (7 tus). So, if we need to consider the time of the information required or released by an activity, we got to extend the definitions given in Section 2.1.

Definition 4 (*require information at t*). If A is an activity then Relation *require* (A, x, t) represents that execution of activity A requires information x at intermediate time t.

Definition 5 (*release information at t*). If A is the same activity, Relation *release* (A, x, t) represents that execution of activity A generates information x at intermediate time t.

Definition 6 (*information constraint*). Applied in between two activities (A_1, A_2): if A_2, requires information x at time point t_{req} and *if* A_1 *releases information x at time* t_{rel} then there exist (from Definitions 4 and 5) two relations: *require* (A_2, x, t_{req}) and *release* (A_1, x, t_{rel}). *It means* information constraint on activity A_2 from A_1, can be written as *constraint* $(A_2, A_1, x, t_{req}, t_{rel})$.

Definition 7 (constraint satisfactions). If an information constraint $(A_2, A_1, x, t_{req}, t_{rel})$ is satisfied, iff exists *require* (A_2, x, t_{req}), *release* (A_1, x, t_{rel}) and $t_{req} \geq t_{rel.}$ A_2 and A_1 is partially overlapped in such a way that intermediate time points of A_2 t_{req} and $t_{rel.}$ of A_1 do not normally occur at the beginning or at the end of activities (A_2, A_1).

According to Definition 6, the information constraints imposed on the activities shown in Figure 2 are listed in Table 3. There are two information constraints imposed on activity A_4, two on A_5, one on A_6 and two on A_7, but no constraint on A_1, A_2, and A_3.

To shorten the design cycle, the idea proposed here is to increase the time overlaps of the activity set in the execution plan so that information constraints are satisfied as early as possible during the time-driven workflow process. The time-driven execution plan semantic is to start an activity when a prior activity is partially finished in order to provide the required information for this activity to proceed. It is not essential to wait for the prior activity to be completely finished. In this scenario, information constraints can be applied at any execution point from timing perspective: at start point, intermediate points, or at end point of an activity.

Considering the information constraints shown in Table 3, the following inferences can be drawn:

(a) *A_1, A_2, and A_3 can be started to execute simultaneously* (*complete overlap – MOC* $= 1$), *when the process is started* (*as shown in Figure* 2, $t = 0$);
(b) *A_4 can be started to execute as soon as A_1 is finished since constraint* $(A_4, A_1, x_1, S(A_4), E(A_1))$ *is satisfied at that moment* $(t = 2)$, *which is different from the common execution semantics*.
(c) *Following the common execution semantics, A_4 need to wait another 4 (tus) for the completion of A_2* $(t = 8)$. *Hence, following the constraint driven execution semantics* (*execution plan* (2)), *A_4 is started* 4 (*tus*) *earlier than following the common execution semantics* (*execution plan* (1)).

The workflow execution plan based on the information constraint semantics is presented as execution plan (2) in Figure 3. From Figure 3, we can find that A_4 is paused in the middle of the execution (where, $t = 6$) because the second constraint *constraint* $(A_4, A_1, x_2, S(A_4) + 4, E(A_2))$ is not satisfied at that moment $(t = 6)$.

So, A_4 needs to wait 2 (tus) until information x_2 becomes available. This means the activity duration of A_4 is prolonged 2 (tus).

Table 4 shows the d_i (lead time), MOCs, and T_i for the execution plan (2). Through comparing the execution plan (1) and (2) by comparing Table 4 and Table 2, we can find that, the whole execution cycle is shortened 11 ($=28-17$) (tus) by following the information constraint driven execution semantics. This is a dramatic improvement. The benefits come from the time alignments using overlaps among A_4 and A_2 and A_5 and A_4. In Execution plan (2), A_4 is started 4 (tus) earlier and A_5 11 (tus) earlier. However, this introduces some side effects: A_4 are paused for 2 (tus) in the middle of the execution, that is the lead time for A_4 is prolonged from 7 (tus) to 9 (tus).

If you are not allowed to pause an activity in the middle of its execution, rearranging the activity execution could eliminate the waiting phase. In this example, to eliminate the waiting phases of the execution for A_4, A_4 should be started at $t=4$ rather than at $t=2$. This corresponds to execution plan (3) as shown in Figure 3. The d_i, MOCs, and T_i for execution plan (3) are shown in Table 5. From Tables 4 and 5, we can see that the total time it takes to finish the execution plan (3) is still the same as execution plan (2), that is 17 (tus).

In this paper, a constraint monitoring process is proposed to deploy the information constraints during an execution plan. Such a monitoring process can also be used to control the timing points (start and end points) of an activity while aligning and scheduling the related constraint satisfaction events. For example, to eliminate the waiting phase, the monitor can delay the event dispatching even when a constraint is satisfied. We will discuss this in the following section.

Table 3. Information constraints for process shown in Figure 1.

Activity	Information Constraints
A_1	NULL
A_2	NULL
A_3	NULL
A_4	$(A_4, A_1, x_1, S(A_4), E(A_1))$; $(A_4, A_3, x_3, S(A_4)+4, E(A_3))$
A_5	$(A_5, A_2, x_2, S(A_5), E(A_2))$; $(A_5, A_4, x_4, S(A_5)+3, S(A_4)+4)$
A_6	$(A_6, A_4, y_4, S(A_6), E(A_4))$
A_7	$(A_7, A_5, x_5, S(A_7), E(A_5))$; $(A_7, A_6, x_6, S(A_7), E(A_6))$

NULL denotes no information constraint on the activity; $S(A)$ denotes the start time point of activity A. And $E(A)$ denotes the end time point of an activity A.

4. Constraint-driven Workflow

4.1 Rule-based Workflow Model

Workflow technology is a key technology to support process execution and management. A workflow system automates a workgroup process and enables information and activity to be routed among the participants. A collection of these rules is called process definition or workflow model. Usually, besides describing sequence of the activity executions, a workflow model concerns other aspects, including Roles, Workflow Relevant Data, and so on. The Workflow Management Coalition (WfMC) has established some workflow-related standards including workflow meta-model and Workflow Process Definition Language WPDL [24].

Many researches and systems use event driven mechanism to support workflow execution. Hence, Event-Condition-Action (ECA) rule is widely used as the description of activity execution sequence [5,7]. The form of an ECA rule is shown as Figure 4. When Event ***e*** occurs, if the Condition ***c*** is satisfied then the Action *a* is taken. The execution sequence of the activities,

Table 4. The d_i, MOCs, and T_i for execution (2).

a_i	$a_1(A_1)$	$a_2(A_2)$	$a_3(A_3)$	$a_4(A_4)$	$a_5(A_5)$	$a_6(A_6)$	$a_7(A_7)$
d_i	2	4	8	9	9	2	4
ts_i	0	0	0	2	4	11	13
$c_i=1-(ts_i-ts_{i-1})/d_{i-1}$	–	1	1	3/4	7/9	2/9	0
T_i	2	4	8	11	13	13	17

a_i is ith activity in the arranged A-set which satisfies Equation (4). A_i denotes the activity A_i shown in Figure 1.

Table 5. The d_i, MOCs, and T_i for execution plan (3).

a_i	$A_1(A_1)$	$a_2(A_2)$	$a_3(A_3)$	$A_4(A_4)$	$a_5(A_5)$	$a_6(A_6)$	$a_7(A_7)$
d_i	2	4	8	7	9	2	4
ts_i	0	0	0	4	4	11	13
$c_i=1-(ts_i-ts_{i-1})/d_{i-1}$	–	1	1	3/4	1	2/9	0
T_i	2	4	8	11	13	13	17

a_i is ith activity in the arranged A-set which satisfies Equation (4). A_i denotes an activity shown in Figure 1.

shown in Figure 1, can be described as a couple of ECA rules shown in Table 6.

The ECA rules, shown in Table 6, represent the activity execution sequences following the typical (common) workflow execution semantic as execution plan (1) discussed above. The related information dependencies among these activities are not represented by these rules. In fact, the ECA rules describe only the control flow but not the data flow. The constraints imposed by these ECA rules do guarantee that the inherent information constraints will be satisfied, but they are often too restrictive. As we have discussed above, it is not possible to exploit a higher level of execution concurrencies due to partial overlaps by following this execution semantics.

To support information constraint-driven execution semantics, it is important that we reformulate the ECA rules to contain the information constraints. However, it is not good to change the form of the rules that describe a workflow. Otherwise, the rule interpreter implemented within a workflow engine needs to be reconstructed, which in turn, may be difficult to implement in most existing workflow management systems. In order to keep the workflow engine unchanged, one simple solution is to add a new event type called SAT(A) to the current ECA based workflow model. Event SAT(A) means all the information constraints imposed on Activity A can be satisfied based on the time estimation in an execution plan

ON	Event	*e*
WITH	Con	c
DO	Action	*a*

Figure 4. ECA form.

Table 6. ECA rules.

Rule 1	ON Event S DO Action ST(A_1); ST(A_2); ST(A_3)
Rule 2	ON Event END(A_1) AND END(A_3) WITH NOT IsNull(x_1) AND NOT IsNull(x_3) DO Action ST(A_4)
Rule 3	ON Event END(A_2) AND END(A_4) WITH NOT IsNull(x_2) AND NOT IsNull(x_4) DO Action ST(A_5)
Rule 4	ON Event END(A_4) WITH NOT IsNull(y_4) DO Action ST(A_6)
Rule 5	ON Event END(A_5) AND END(A_6) WITH NOT IsNull(x_5) AND NOT IsNull(x_6) DO Action *ST*(A_7)

Event *S* means the event of starting the process, END(A) denotes the event of the completion of activity *A*. Action ST(A) means start to execute *A*. IsNull(x) is a built-in function to check whether the variable x is NULL to guarantee it is valid.

if the activity is started when this SAT(A) event occurrence. A SAT(A) event is an event to initiate an activity execution. In fact it can also be viewed as a schedule event. Event SAT(A) can be created and sent by a constraint monitor. The latter will be discussed in the following section. When a SAT(A) event is created and sent, it means that some partial information constraints has been already satisfied. The remaining information constraints are considered to be satisfied without delays in accordance with the time estimations. So, for the constraint monitor, one of its functionality is to find the appropriate time point to initiate the activity A, that is, release the SAT(A) event.

For example, referring to execution plan (3), shown in Figure 3, we will discuss how to determine the time point for starting activity A_4 as shown in Table 3. There are two constraints on activity A_4, which are:

C_{41} : *constraint* (A_4, A_1, x_1, $S(A_4)$, $E(A_1)$)
C_{42} : *constraint* (A_4, A_2, x_2, $S(A_4)+4$, $E(A_2)$)

Constraint C_{41} denotes that, to start activity A_4, information x_1 must be available, which means C_{41} must be satisfied because information x_1 is needed at the starting point $S(A_4)$. So, C_{41} is the initial information constraint, but C_{42} is not. However, the selection of time point $S(A_4)$ must ensure that C_{42} will be satisfied without waiting for information x_2 in the middle of activity execution. The problem to find such a starting time point for activity A_4 under these two constraints can be represented mathematically as:

$$\begin{aligned} &\text{Minimize } S(A_4) \\ &\text{Subject to } S(A_4) \geq E(A_1) \end{aligned} \tag{9}$$

$$S(A_4) + 4 \geq E(A_2) \tag{10}$$

According to Equation (9), the minimize $S(A_4)$ should be $E(A_1)$ ($t=2$), but according to Equation (10), to eliminate the waiting phase for information x_2, it is better that activity A is started at $t=4$ ($=E(A_2)-4=8-4=4$). Hence, we can find that it is a Constraint Satisfaction Problem (CSP) to determine a time point for starting an activity A, which can be formulated as:

$$\begin{aligned} &\text{Minimize : } S(A_4) \\ &\text{s.t.} \forall_{c \in C(A),\, treq,\, trel \in c} \exists t_{req} \geq t_{rel} \\ &(C(A) \text{ is the information constraint set of activity } A) \end{aligned} \tag{11}$$

By adding the new event type SAT(A), some ECA rules can be changed to the new ones to support the constraint-driven execution semantics. By considering the constraints on activity A_4, shown in Table 3, Rule 2 shown in Table 6 is rewritten as rule 2 in Table 7.

Table 7. Rewritten ECA rules for Rules 2 and 3.

Rule 2	ON Event SAT (A_4) WITH NOT IsNull (x_1) DO Action ST (A_4)
Rule 3	ON Event SAT (A_5) WITH NOT IsNull (x_2) DO Action ST (A_5)

The event field END(A_1) AND END(A_2) is changed into SAT(A_4). One will argue that it is not necessary to change the composite event END(A_1) AND END(A_2) to SAT($A4$) since both x_1 and x_2 are released in the end of the executions of A_1 and A_2 separately. In a sense, it is right. But, to eliminate the waiting phase, A_4 is not started when x_1 is released or A_1 is completed, but 2 (tus) after that. Traditional workflow engine cannot control the occurrence of events. However, through the constraint monitor, the occurrence of event SAT(A_4) can be controlled since all SAT(A) events are created and sent by the constraint monitor rather than workflow engine or workflow client application.

One can rewrite other ECA rules shown in Table 6 by following the same idea used earlier to change Rule 2. But, not all the rules are needed to be changed. In fact,

- Only Rules 2 and 3, shown in Table 6, need to be rewritten, which are shown in Table 7.
- It is not necessary to change Rule 1 since no information constraint is related to Rule 1.
- For Rule 4, no change needed because only an information constraint is imposed on A_6 and it can be satisfied only when END (A_4) occurs. Event END (A_4) is the same as SAT(A_4).
- For Rule 5, for the same reason as Rule 4, no change is needed.

So, what conclusion can be drawn about determining which ECA rules should be changed? The answer is:

Proposition 2 Only when there exists an activity information constraint $(A, A_1, x, t_{req}, t_{rel})$ imposed on activity A where $t_{req} \neq S(A)$ or $t_{rel} \neq E(A_1)$, the corresponding ECA rule which specify the start action of activity A has to be changed.

4.2 System Architecture

The workflow system architecture that supports the information constraint-driven execution semantics is shown in Figure 5. This architecture is extended from WfMC workflow reference model [22] and common ECA-based workflow system architecture.

The project management tool is used to set up the project and make the process execution plan, including activity sequences, activity duration estimation, activity dependencies and so on. An example of this tool is Microsoft Project.

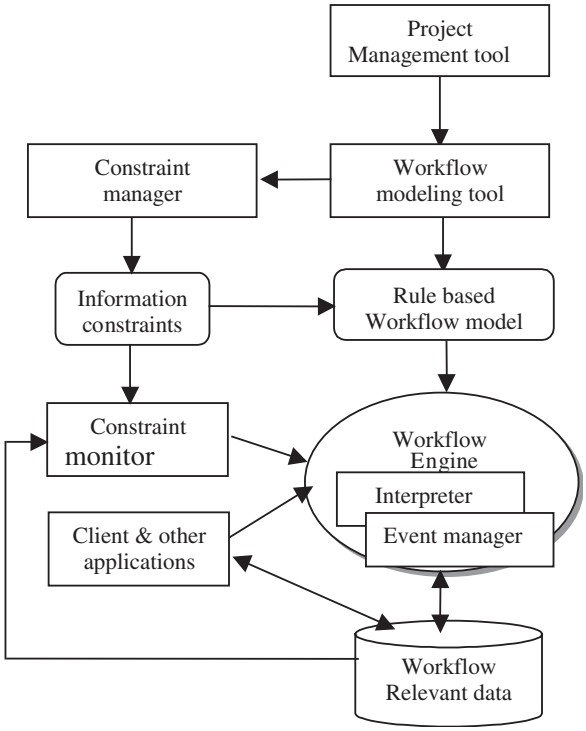


Figure 5. Constraint-driven workflow system architecture.

The workflow-modeling tool imports the project model and adds additional information to make the process executable. For example, roles, resources, and the application information required to execute the activities are assigned to the activities in workflow modeling phase. In order to support constraint-driven workflow execution semantics, the information constraints and the time points – the activities require or release them – should be specified in the workflow model. ECA rule-based workflow definition can be generated from the graph-based workflow-modeling tool.

The constraint manager will extract the information constraints from the workflow definition according to the activity dependencies and the specified information; especially the information required and released setting. The constraint manager manages these information constraints and rewrites the related ECA rules by using the method we presented above.

The workflow engine enacts the ECA rules after the workflow model is instanced. The event manager, which is an important part of the engine, receives the "END(A)" events from the client applications or workflow engine itself as well as the "SAT(A)" events from the constraint monitor. The interpreter, the core of the workflow engine uses the events to interpret the ECA rules and to initiate the execution of the activities.

During workflow execution, an activity can be executed by a client or a server application. The client application (or a workflow engine) updates (including release) the workflow relevant data (including activity sharing information). The constraint

monitor monitors this updated information. When some related information is released (not all update operations are "release" operations), the constraint monitor checks the information constraints that are related to the released information. The related constraints are set to be satisfied, if the required information is already available. The constraint monitor will calculate all the information constraints imposed on this related activity and determines whether it is time to release a "SAT(A)" event and send it to the workflow engine to initiate this related activity.

5. Discussion and Conclusion

To develop a better execution plan, it is essential that concurrency is exploited as early as possible even with partial overlaps of the activities. This ensures that intermediate time points are considered in the execution plan during the concurrent product design. As such this paper focuses on two important questions (a) When does the activity require the information from its upstream activities and (b) when will the required information be available?

An information constraint-driven execution plan semantics is proposed to relax the time restrictions on the information constraints. This enables that a dependent activity set be executed earlier in the process even if partial information for the "information constraints" are available.

Generally, two basic situations occur in a concurrent design process, which is the subject of this research:

1. Some information is released at the intermediate time point in the execution plan and not at the end of an activity;
2. Some information is required at the intermediate time point in the execution plan and not at the beginning of an activity.

For situation (1), one will argue that why the initial breakdown of the design activities has not been based on the availability of information at the end of that activity. In reality, especially for supporting CE, this situation is becoming common. For example, when designing a part that is made up of several curve pipes and other elements, when the diameter of one of the pipes and the radius of its arc are determined, but the design activity of the whole part is still in the progress of design, the downstream DFM activity for checking the manufacturing of the pipe can be started to check whether the diameter and the radius corresponds with the manufacturing specification. If the selection of the diameter or radius is incorrect, the designer can receive the modification suggestion from the DFM activity immediately. Apparently, it is not reasonable to break this design activity down into several ones according to how many pipes it needs. For situation (2), this is very common for large, long term and complex activity, which requires different information from several upstream activities in different phase.

The proposed execution plan semantics can dramatically shorten the design cycle time if the activity duration and the time when the information required and released are accurately estimated in the initial project-planning phase. This time-driven execution plan semantics is very suitable for complex workflow processes and large set of long activities that cannot be easily broken down into smaller pieces. The concept of time and constraint driven execution plan semantics can be extended to in scheduling workflow execution in a WfMS.

References

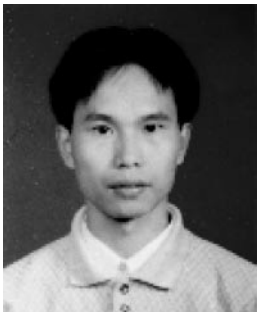
1. Aversano, L., Canfora, G. and De Lucia, A. et al. (2001). Business Process Reengineering and Workflow Automation: A Technology Transfer Experience, *Journal of Systems and Software*, Available on line on 24 Oct., 2001.
2. Bitzer, S.M. and Kamel, M.N. (1997). Workflow Reengineering: A Methodology for Business Process Reengineering Using Workflow Management Technology, In: *Proceeding of the 30th Hawaii International Conference on System Sciences*, pp. 415–426.
3. Chlebus, E., Cholewa, M. and Dudzik R., et al. (1998). CAx Application for Process Oriented Concurrent Design, *Journal of Materials Processing Technology*, **76**(1–3): 176–181.
4. Eppinger, S.D. (2001). Innovation at the Speed of Information, *Harvard Business Review*, **79**(1): 149–158.
5. Geppert, A., Tombros, D. and Dittrich, K.R. (1998). Defining the Semantics of Reactive Components in Event-driven Workflow Execution with Event Histories, *Information Systems*, **23**(3–4): 235–252.
6. Gayretli, A. and Abdalla, H.S. (1999). An Object-oriented Constraints-based System for Concurrent Product Development, *Robotics and Computer-Integrated Manufacturing*, **15**(2): 133–144.
7. Goh, A, Koh, Y.K. and Domazet, D.S. (2001). ECA Rule-based Support for Workflows, *Artificial Intelligence in Engineering*, **15**(1): 37–46.
8. Huang, G.Q., Huang, J. and Mak, K.L. Agent-based Workflow Management in Collaborative Product Development on the Internet, *Computer-Aided Design*, **32**(2000): 133–144.
9. Kuo, T.C., Huang, S. H. and Zhang, H.C. (2001). Design for Manufacture and Design for '*X*': Concepts, Applications, and Perspectives, *Computers & Industrial Engineering*, **41**(3): 241–260.
10. Kovacs, Z., Le Goff, J.M. and McClatchey, R. (1998). Support for Product Data from Design to Production, *Computer Integrated Manufacturing Systems*, **11**(4): 285–290.
11. Krishnan, V., Eppinger, S.D. and Whitney, D.E. (1997). A Model-Based Framework to Overlap Product Development Activities, Management Science, **43**(4): 437–451.

12. Prasad, B. (1999). Enabling Principles of Concurrency and Simultaneity in Concurrent Engineering, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **13**(3): 185–204.
13. Prasad, B., Wang, F. and Deng, J. (1998). A Concurrent Workflow Management Process for Integrated Product Development, *Journal of Engineering Design*, **9**(2): 121–135.
14. Shih, H.M. and Tseng, M.M. (1996). Workflow Technology-based Monitoring and Control for Business Process and Project Management, *International Journal of Project Management*, **14**(6): 373–378.
15. Spinner, M. (1989). Improving Project Management Skills and Techniques, *Englewood Cliffs*, NJ: Prentice Hall.
16. Steward, D. (1981). *System Analysis and Management: Structure, Strategy and Design*, New York: Petrocelli Books.
17. Steward, D. (1991). Planning and Managing the Design of Systems, In: *Proceedings of Portland International Conference on Management of Engineering and Technology*, Portland, Oregon, USA, 27–31 October.
18. Ulrich, K.T. and Eppinger, S.D. (1994). *Product Design and Development*, New York: McGraw-Hill Inc.
19. Steward, Donald, V. (1981). The Design Structure System: A Method to Managing the Design of Complex Systems, *IEEE Transactions on Engineering Management*, **28**(3): 71–74.
20. Prasad, B. (1996). *Concurrent Engineering Fundamentals, Volume I: Integrated Product and Process Organization*, Upper Saddle River, New Jersey: PTR Prentice Hall.
21. Prasad, B. (1997). *Concurrent Engineering Fundamentals, Volume II: Integrated Product Development*, Upper Saddle River, New Jersey: PTR Prentice Hall.
22. WfMC (Workflow Management Coalition) (1995). WfMC Workflow Reference Model, *WFMC-TC00-1003* (issue 1.1), 19 Jan, 1995.
23. WfMC (Workflow Management Coalition) (1999a). Workflow Management Coalition Terminology & Glossary, *WFMC-TC-1011*(*Issue 3*.0), Feb., 1999.
24. WfMC (Workflow Management Coalition) (1999b). Interface 1: Process Definition Interchange Process Model, *WFMC-TC-1016-P* (version 1.1, Official Release), 29 Oct., 1999.

Dr. Brian Prasad

Brain Prasad is the Chief Knowledge Officer at Spec2-Market Solutions, in Tustin, CA, author of several books, and managing editor of the most premier Int. J. of Concurrent Engineering: Research & Applications. Prior to joining Spec2Market, he was a Visiting Professor at California Institute of Technology (CAL-Tech), Pasadena. He is a senior executive consultant (UGS), and on the Board of Advisors to many large fortune 500 companies including GM, GE and EDS. He has secured and managed large funded research from many (Government and Private) sources. He is a syndicated columnist for publications including Value-based Management, Business Process Management, Journal of Manufacturing Systems, and Industrial Knowledge Management, and taught at UCI and California State University Fullerton, School of Engineering. He holds many adjunct faculty positions at Oakland University, Rochester; Wayne State University, Detroit; West Virginia University, Morgantown; to list a few. During 2001, Dr. Prasad served as the Director of the Engineering, Information Technology and Sciences unit of the University of California at Irvine Extension (UNEX), CA. Dr. Prasad holds a PhD degree in Mechanical and Aerospace Engineering from Illinois Institute of Technology, Illinois, Chicago. He also graduated from the Stanford University, School of Engineering with a Degree of Engineering in Applied Mechanics (now a Division of Mechanical Engineering), California. He received a Master (MS) degree from Indian Institute of Technology, Kanpur and a BE Degree from Bihar College of Engineering, Patna, both from India.

Jinmin Hu



Jinmin Hu is a senior software architect at Kingdee Software Co. Ltd, one of the biggest ERP vendors in China. He is also a part-time researcher at CIT Lab, Shanghai Jiao Tong University, China. He worked as a postdoctoral researcher at University of Twente, the Netherlands in 2001 and 2002. His research interests are in workflow management system, concurrent engineering and software engineering. He received an MS in automatic control from Hunan Univeristy and a PhD in computer science from Shanghai Jiao Tong University. His email address is hujinmin@ yahoo.com.cn

Jianxun Liu

Jianxun Liu is now an associate professor in CS department of Hunan University of Science and Technology, China. He received his MSc and Ph.D degree in computer science from Central South University of Technology in 1997 and Shanghai Jiao Tong University in 2003 separately. His current research interests include Electronic Commerce, Workflow, Agent and XML.