Association for Information Systems
AIS Electronic Library (AISeL)

PACIS 1993 Proceedings

Pacific Asia Conference on Information Systems (PACIS)

December 1993

PM-Net: A Software Management Represenational Model

Kuen-Ching Lee
National Sun Yet-sen University

Hsin-Hui Lin
National Sun Yet-sen University

Iuan-Yuan Lu
National Sun Yet-sen University

PM-Net : A Software Project Management Representation Model

Kuen-Ching Lee[1], Hsin-Hui Lin[2] and Iuan-Yuan Lu[1]
[1] Institute of Business Management, [2] Institute of Information Management
National Sun Yat-sen University, Kaohsiung, Taiwan, R. O. C.

Abstract - A model called PM-Net for representing and monitoring the software development process is presented. This model provides information for progress management, as well as information of project status at different levels of detail, for the benefit of project managers. This model emphasizes bottom-up data collection and top-down information inquiry functions. The Data Flow Diagram (DFD) and Work Breakdown Structure (WBS) techniques are utilized for construction of a hierarchical structure of the software development process. This process can be viewed as a set of activities; each activity can be viewed as a set of subactivities, and each subactivity can be viewed as a set of tasks. Managers can make an inquiry regarding project progress information at process level, activity level, subactivity level or task level, depending on the degree of details they require. The manager can also change the control state operators at the lowest level of activity which will enable a transition for a change of process status at the upper-level activity. This ensures that any large-scale software development project based on PM-Net will be able to represent the most recent project progress status at any time in the project's running duration, so that managers can monitor the project progress more efficiently.

Index Terms : DesignNet, Petri Net, project progress, project schedule, software project management

Introduction

1. The Main Problems of Software Project Control

Project management includes a mixture of people, resources, systems, and techniques required for carrying out a project to successful completion [6],[13]. The goal of project control is to accomplish the project before the scheduled dead-line, within the budget, meeting the specification, and also efficiently utilizing the resources. However, cost overrun, late completion, staff turnover and low quality often occurs in software development projects, especially in large-scale software development operations [6],[14],[15]. Correctly estimating software cost could become a solution to the problem. However, most software projects have the following features which result in difficulties in correctly estimating software cost : (1) the unique requirements of each software project, (2) the uncertainty of software size estimation, and (3) the uncertainty of user requirements.

Relying on software cost estimation is not enough since it is far from accurate and the estimated costs may possibly influence the final project cost [1],[2],[3]. A method is therefore required here which assists the managers in monitoring the actual project progress and in managing the development process.

The lack of an appropriate model for the development of large-scale software, has recently been the focus of a great deal of discussion and concern [4],[9],[11]. Lack of control of software project schedule and cost seems to be an international concern. The development of a model for representing the project progress and resources utilization therefore becomes important.

2. The Inadequacies of the Traditional Project Management Models

The Gantt Model [10], The Critical Path Method (CPM) [12] and The Program Evaluation and Review Technique (PERT) [20] are three of the traditional project planning and control models. All of these approaches, Gantt, CPM, and PERT, focus on the scheduling of activities. However, this simple formulation does not capture important characteristics of software projects, with the consequence that the schedule often becomes out-of-date after the project starts running. In terms of software project schedule control, the traditional models have the following deficiencies of :

(1).None of these models provide the information that would permit the manager to analyze and draw conclusions regarding about the progress of activities [14];
(2).Current models are inadequate for representing the hierarchical relationship of activities and subactivities as an integral system component;
(3).Activity dependencies do not include the notion of boolean conditions, even though they handle predecessor activities[14];
(4).Current models are unable to represent the reschedule when a completed activity is being reactivated;
(5).Current models are not capable of providing the information to the manager when an activity is activated before all of its prior activities have been completed;
(6).Current models are also inadequate for representing the criteria that trigger the start of an activity.

Review of Network Model

The formal definitions and functions of Petri net, UCLA model and DesignNet are first introduced here in this section. The inadequacies of these models in software project management are also discussed here.

1. Petri Net Model

The Petri net is an abstract model utilized for describing and analyzing information and control flow in asynchronous concurrent system [17],[18],[23]. A Petri net C is formally defined as a four tuple $C = (P,T,I,O)$. $P = \{p_1, p_2, ...,p_n\}$ is a finite set of places, $n \geq 0$. $T = \{t_1, t_2, ...,t_m\}$ is a finite set of transitions, $m \geq 0$. The set of places and the set of transitions are disjoint, $P \cap T = \emptyset$. I is an input function and O is an output function. Directed arc from the places to the transitions and from the transitions to the places are represented by the input and output functions.

The input function I is a mapping from a transition t_j to a collection of places $I(t_j)$, known as the input places of the transition. The output function O maps a transition t_j to a collection of places $O(t_j)$, known as the output places of the transition. A marking vector $\mu = (\mu_1, \mu_2,...,\mu_n)$ gives the number of tokens in a place for each place at a particular time. The number of tokens in place p_i is μ_i, $i = 1,2,...n$. A Petri net with a marking μ becomes a marked Petri net, $M = (P, T, I, O, \mu)$. Examples of a graph representation of a marked Petri net are shown in *Fig. 1*.

A Petri net graph consists of two types of nodes. A circle "O" represents a place, where one or more tokens(represented by small dots "• ") can reside; a bar " | " represents a transition, which can be fired to move tokens from inputs to outputs.

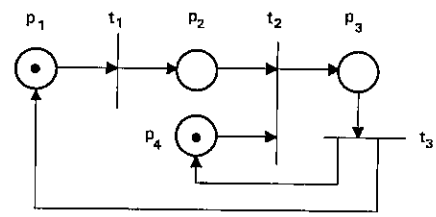

Fig. 1 An example of Petri net

A Petri net executes by firing a transition. A transition fires by removing tokens from its input places and creating new tokens which are distributed to its out places. A transition may fire if it is enabled to do so. A transition is enabled to do so if each of its input places have at least as many tokens in it as *arcs* from the place to the transition. Two tokens are shown as an example in *Fig. 1* to be initially placed in p_1, p_4; transition t_1 will then be fired and token in p_1 will be moved to p_2. After p_2 has an token, then t_2 will be fired, and two token in p_2, p_4 will move to p_3. Once p_3 has two tokens, t_3 will be fired and two tokens in p_3 will be moved to p_1, p_4. Firing a transition can be considered as the happening of an event. It may change the state of the system, causing some conditions to cease holder and others to begin to hold.

A place is assumed here to be utilized for representing a planned activity and a token inside it means that this activity is currently activated. Some deficiencies toward applying the Petri net to software project management are :

(1) A token in Petri net graph is only a boolean value conditions. The project manager cannot expect how soon that an activity will be completed or started, and cannot judge whether the project's progress has been delayed or not;

(2) The execution of a Petri net is nondeterministic[14]. If more than one transition is enabled at any time, it is not predictable which one will fire first. This makes it more complicated to analyze the properties of a system;

(3) If an executing activity has to be suspended for its responsible staff turnover, Petri net cannot move the token in that place out, such that the manager is still to be informed that the activity is to continue on executing;

(4) Petri net restricting a transition t_j will be fired when each of its input places has at least many tokens in it. But sometimes, for avoiding the project schedule delay, the manager probably has to fire the transition to activate the activity a_i that $a_i \in O(t_j)$ before each of $I(t_j)$ has token in it;

(5) Petri net cannot help the project manager in distinguishing out an executing activity whether that activity is a new creating or re-executing.

2. UCLA Graph Model

UCLA is the couple biologic graph model of computation [8],[18]. In this model systems are represented by a graph with complex directed arcs. A UCLA model is a six-tuple $\mathbf{C} = (V, A, L, Q, S, F)$. Where $\mathbf{V} = (v_1, v_2, ..., v_r)$ is a set of vertices

$\mathbf{A} = (a_1, a_2, ..., a_s)$ is a set of arcs.

$\mathbf{L} = (L^-, L^+) : V \rightarrow \{*, +\}$ is the input(L^-) and output(L^+) logic mapping for each vertex.

$\mathbf{Q} = (Q^-, Q^+), V \times A \rightarrow N$ is the output(Q^-) and output(Q^+) degree of each arc-vertex pair

$\mathbf{S} \in \mathbf{A}$ is the start arc, $\mathbf{F} \in \mathbf{A}$ is the final arc.

UCLA is a kind of model like PERT. It can also be transferred to an equivalent model of Petri net. UCLA utilizes combinative logic to control the sequencing of operations at the nodes. Tokens are required on each input arc to enable an operation; that is, if the input logic of a node is *AND*(*). For *OR*(+) logic, tokens are only required on some one input arc. For *AND* output logic, tokens are placed on all output arcs, while for *OR* logic, tokens are placed on any one output arc.

An example of an UCLA graph is shown in *Fig. 2*. The logic of each arc-node pair is notably marked on the graph as either * or + logic. In this example node *a* can fire whenever arc *S* has a token. When node *a* fires, it removes token from arc *S* and puts tokens on both arc *A* and arc *B*(*AND* logic). Node *c*, on the other hand, will place a token on either arc *D* or arc *E*(*OR* logic). Node *e* is enabled whenever there is one token on arc *C* and one token on arc *F*. Also, node *g* is enabled whenever arc *G* or arc *H* has a token.

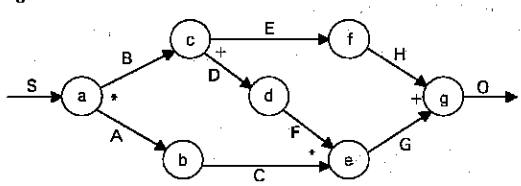

Fig. 2 An example of UCLA graph model

For a project management model, UCLA has same deficiencies like as in Petri net and PERT. Its notation of *AND*(*)/*OR*(+) logic could, however, be transferred to a project management model named PM-Net that is to be proposed here later on, such that it could enhance the functions of PM-Net model.

3. DesignNet Model

DesignNet is a hybrid model, which utilizes *AND/OR* structure operators for description of the work breakdown structure and Petri net notation used for representing the dependencies among activities, resources, product, was proposed by *L. C. Liu* and *E. Horowitz* in 1989 [14]. The waterfall model has also be mapped into DesignNet by them. *AND/OR* graphs are used in artificial intelligence for modeling a task in terms of a series of goals and subgoals [21]. A DesignNet graph D is defined as five-tuple : $D = (P,T,S,I,O)$. Where P is a set of places; $P = \{ P_a, P_r, P_p, P_s \}$. Four place types are defined including activity P_a, resource P_r, product P_p, and status report P_s. T is a set of transition; $T = \{ T_s, T_f \}$, T_s is a transition which starts an activity and T_f is a transition after an activity is completed. S is a set of structure operators; $S = \{ S_o, S_a \}$, S_a is an "*AND*" structure operator; S_o is an "*OR*" operator. I is a set of input arcs; $I = \{ I_s, I_f, I_a, I_o \}$; I_s represents the set of input arcs for a transition T_s, from a place which is of type product or resource; I_f represents the set of input arcs for a transition T_f, from a place which is of type activity or status; I_a (I_o) represents the set of input arcs for connecting a place node to an operator $S_a(S_o)$. O is a set of output arcs; $O = \{ O_s, O_f, O_a, O_o \}$, $O_s(O_f)$ represents the set of output arcs for a transition $T_s(T_f)$; O_a (O_o)represents the set of output arcs for connecting an operator $S_a(S_o)$ to a place node. A DesignNet consists of a set of *places*, a set of *structure operators*, and a set of *transitions*. Structural operators connect places of the same type on two adjacent levels where the lower level places are the decomposition of the higher level place. The hierarchy resulting from the connection of structural operators is the WBS of the project. Prerequisite conditions (products and resources required) before an activity can start and the product generated by an activity are linked to activities by transitions.

A DesignNet functions by firing transitions. Unlike the firing of a transition in Petri nets, which causes tokens to be moved from their input places to their output places, the transition firing in a DesignNet is a nonvolatile process. Each transition can be considered as an executable procedure for the sake of handling the transition firing operation. The procedure associated with the transition is to become executed whenever a new token is created in the input place of a transition. This procedure would check against all the input places. The input places, if having active tokens, performs the firing operation by setting input tokens to a consumed state and creating new tokens in output places through instantiation.

DesignNet, despite potentially being a powerful model for software project control, has some deficiencies as follows :

(1). If an activity have several products in its output places, DesignNet cannot display how many of its output products have been completed until all output products are completed. A manager can therefore not know the percentage of completion until it is fully completed;

(2). There are cases such as readjustment of the schedule that an activity has to be activated before each of its input places have at least one active token in it. But according to the transition firing rule, DesignNet cannot handle this condition which often occurs during software development projects;

(3). Different levels of managers require having different kinds of information about project progress, but DesignNet only provides one kind of project progress information;

(4). A current activity will sometimes be forced to become suspended because of staff turnover, requirement specification change, design specification change, etc.. DesignNet does not take into account this possibility;

(5). In DesignNet, the token for a completed product will be changed from "active" to "consumed" state once one of its output activity place has an active token in it, such that the product will be prohibited from activating other output activity places. A product can, however, become the input place of more than one activity. This feature of DesignNet does not reflect the actual conditions of software development.

The PM-Net Model

The basic PM-Net concept is first introduced in this section and then it is given a detailed explanation. The implications of PM-Net for software project control are then next discussed here. Certain features of the PM-Net model are then finally clarified.

1. The Basic Design Concept of PM-Net

(1) Providing Information that can Meet Management Requirements

Managers at different levels concern about the project's progress for a large-scale software project. Top-level managers are concerned about the percentage of completion and the percentage of budget consumption. Division managers want to know what phase of work has been completed, what phase of work is being performed, and whether the project schedule has been delayed or not. Project managers must have more detailed information regarding the whole project than what is necessary for the division manager. Bottom-level managers want to know the actual progress of a subactivities or a task. To meet the requirements of different levels of management, we adopted the Data Flow Diagram (DFD) analysis technique [16], [24] instead of the waterfall model [19] to DesignNet. The transition firing rule, the token propagating rule and the token types have been modified here in some places. This modification is to compensate for the limitations of DesignNet.

For a large-scale software development project, the DFD analysis technique can decompose a project into several distinct process, and then each process can become decomposed into a set of activities. An activity can become decomposed into a set of subactivities, and even a subactivity can be decomposed into a set of tasks. Two principles have to be followed whenever decomposing project activities and their output products. These two principles are borrowed from analyzing the business process method [7].

(a). Each of the activities at the bottom level of the hierarchy are permitted to have only one output product. If more than one product exists in its output, then those products should be composed into one, or the activity will become decomposed into smaller units again. This occurs owing to the bottom-level manager being required to have continual access to the status of an activity and its output product for which he is responsible for, so that he can make an accurate assessment of project's progress. Based on the rule of firing transitions, if an activity has two or more output products, the manager cannot judge which one of the products will be completed first until all of its output products have been completed. For example, if a_i is a bottom-level activity in PM-Net, and if products P_{j1}, P_{j2} are the output of activity a_i such that $(P_{j1}, P_{j2}) \in O(a_i)$, then we either have to let $\{P_j\}= \{P_{j1}, P_{j2}\}$, such that $P_j \in O(a_i)$, or let $\{a_i\}= \{a_{i1}, a_{i2}\}$, such that $P_{j1} \in O(a_{i1})$, $P_{j2} \in O(a_{i2})$, and $P_{j1} \notin O(a_{i2})$, $P_{j2} \notin O(a_{i1})$.

(b). Each output product, whatever its level in the hierarchy may be, is permitted to be an output for only one activity. If a product is produced by two or more activities in the model, the product should be decomposed, or those activities should be combined into one activity. For example, if a_{i1}, a_{i2} are two activities in PM-Net, and product $P_j \in O(a_{i1})$, $P_j \in O(a_{i2})$, then either $\{P_j\}= \{P_{j1}, P_{j2}\}$ has to be decomposed here, such that $P_{j1} \in O(a_{i1})$, $P_{j2} \in O(a_{i2})$, and $P_{j1} \notin O(a_{i2})$, $P_{j2} \notin O(a_{i1})$, or a_{i1}, a_{i2} have to be composed to an activity a_i such that $\{a_i\}= \{a_{i1}, a_{i2}\}$, then $P_j \in O(a_i)$.

These above two principles ensure that the representation of software development activities in PM-Net are precise.

(2) An Appropriate Transition Can be Fired and Enabled Whenever an Event Occurs

A transition can be enabled and fired whenever an event occurs at its input places. An activity token can sometimes be generated and become active before all of its input places are active. An executed activity can also sometimes be suspended awaiting for the completion of product or awaiting specific technical resources. The model cannot be limited so that a transition can be fired only when each of its input place has at least one active token in it, as this will induce the model for show a current status delay in the project's progress.

2. PM-Net Structure

A PM-Net is composed of a six-tuple $D = (P, T, I, O, F, \mu)$ where P is a set of *places* that is a union of five possible types : $P = (P_a, P_r, P_p, P_s, P_f)$, P_s is a place of *status operator*, $P_s = \{ss, fs\}$; *ss* is a *success status operator*; *fs* is a *fail status operator*; and P_f is a place of *failure report*. T is the set of *transitions* that are a union of two possibilities, $T = (T_s, T_f)$. I, O is a set of input and output arcs for transitions respectively, $I = (I_s, I_f)$, $O = (O_s, O_f)$, $I(a_j)$ is the input places of activity a_j through transition T_s, $O(a_j)$ is the output places of activity a_j through transition T_f. P_a, P_p, P_r, T_s, T_f, I_s, I_f, O_s, O_f, all have the same meaning as that defined in DesignNet. F is an expression of the input *control state operators* which evaluate whether a transition T_s will be enabled or not. $F = (*, +)$, is a set of operators modified from the UCLA graphs [5],[8],[18],[22]. "*" is an *AND* logic that means active tokens are needed on P_2 has an active token(1*), then the transition T_s will be enabled activated activity a_1. In *Fig. 3b*, if each of P_1 and R_1 has an active token, the transition T_s will be enabled and will initiate an active token at a_1. If each of P_2 and R_1 has an active token, and P_1 does not, the transition T_s cannot be enabled. In *Fig. 3c*, the transition T_s will not be enabled until of its input places has at least one active token in it. In *Fig. 3d*, if R_1 and either two of P_1, P_2, P_3 have active tokens(2*), the transition T_s will be enabled and generate an active token in a_1. The type of control state operator at any bottom-level place in the model can be manually changed throughout the duration of project execution. A marking μ in PM-Net are defined as a mark vector of the number of tokens in a place, its definitions and functions in the model are the same as in DesignNet.

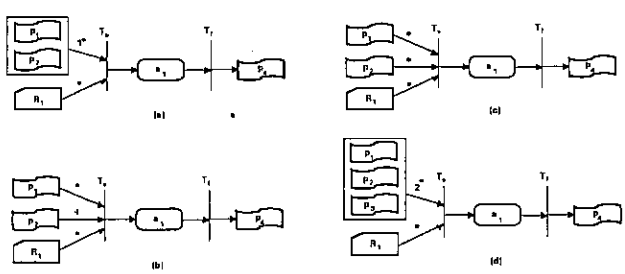

Fig. 3 A control state operator graphic representation

input place to enable the transition T_s, "+" is an *OR* logic that means active tokens aren't needed on that input place to enable the transition T_s in currently. The operators "*", "+" being placed at different place will represent different meanings. For example, in *Fig. 3a*, when each of R_1 and either P_1 or

3. Interconnection of Activities

An interconnection of activities can be defined so as to complete the PM-Net structure at a certain level. The interconnection relation R is a binary relation.

$$R \subset a \times a$$

The actual interconnection of activities is made through product P_k, such that if $(a_i, a_j) \in R$, and if $P_k \in O(a_i)$, $P_k \in I(a_j)$, where $I(a_j)$ means the input places of the activity a_j, through transition T_s, $O(a_i)$ means the output places of the activity a_i through transition T_f, then $O(a_i) \cap I(a_j) \neq \phi$, that is the activity a_i and a_j are connected through the product P_k. The interconnection relations can therefore be defined can be verified here as follows:

$$R_{ij} = \{ (a_i, P_k, a_j) \mid P_k \in O(a_i) \cap I(a_j) \}$$

where $i = 1, 2, ...$, and $j = 1, 2, ...$

If $P_{k1} \in O(a_{i1})$, $P_{k1} \in I(a_{j1})$ and $P_{k2} \in O(a_{i2})$, $P_{k2} \in I(a_{i2})$, and if $\{a_i\}= \{a_{i1}, a_{i2}\}$, $\{a_j\} = \{a_{j1}, a_{j2}\}$, then P_{k1}, P_{k2} will be combined into one product P_k such that $\{P_k\} =\{P_{k1},P_{k2}\}$. To ensure the consistency of the different levels of activities throughout the PM-Net graph, the interconnection relations between different levels activities can be verified here as follows :

First, derived here from the above definition is :

$$R_{i1j1} = \{ (a_{i1}, P_{k1}, a_{j1}) \mid P_{k1} \in O(a_{i1}) \cap I(a_{j1}) \}$$

$$R_{i2j2} = \{ (a_{i2}, P_{k2}, a_{j2}) \mid P_{k2} \in O(a_{i2}) \cap I(a_{j2}) \}$$

Then, obtained :

$$R_{ij} = \{ [a_i, (P_{k1}, P_{k2}), a_j] \mid (P_{k1}, P_{k2}) \in O(a_i) \cap I(a_j) \}, \text{ or}$$

$$R_{ij} = \{ (a_i, P_k, a_j) \mid P_k \in O(a_i) \cap I(a_j) \}$$

where $\{P_k\} = \{P_{k1}, P_{k2}\}$, a_i is the upper level activity of a_{i1}, a_{i2}, and a_j is the upper level activity of a_{j1}, a_{j2}.

4. Definitions and Implications of Component Notation in the Model

(1). Token State Types

A transition firing in a PM-Net is a nonvolatile process, as in DesignNet. A token can be in different states at different places. A token in an activity place can be in one of five possible states : *active*, *suspended*, *complete*, *failure to complete*, *discarded*. In a product place, a token can be in *creating*, *active*, *suspended*, or *discarded* states. In a resource place, a token can be in an *active*, *empty*, or *discarded* state. In a status operator and status report place, a token can be in either an *active* or *discarded* state. A transition T_s at the bottom-level can be enabled if there is one active token in each of its input places whose control state operator is * for *AND* logic, as well as an active token isn't needed in its input places with control state operator which is + for *OR* logic. When a transition T_s is enabled, it will be fired and generate one active token in its output "activity" place. When an activity has an active token, then the transition T_f will be enabled, and its output product place will generate a "creating" token. When the token state in an activity is in "complete" or "suspended" state, the token state in its output "product" place will be in "active" or "suspended" state. A token of a different type in a different place has a different meaning. A token life cycle ranges from "active" or "creating" state to "discarded" state, Since the transition firing is nonvolatile, the number of marking μ_i in a place P_i means that the same number of tokens have been generated.

(2). Token Type Notation

According to the token type definition in the prior section, a token in a different state in a different place has a different meaning. The tokens in different states should therefore be represented by different notations. The definition of token type notation is provided in *Table 2*. By these, a manager can quickly understand the current status of a software development project from the notations of PM-Net represented.

Table 2 The notation of token type

token type	Notation
active	•
creating	.
complete	!
failure to complete	&
suspended	;
empty	;
discarded	⊗

(3). The Priority of Firing a Transition

A transition will be fired in Petri net or DesignNet on the basis of whether all of its input places are in active state or not. No consideration is given to transition firing priority. In the case of PM-Net, more information about project scheduling is hopefully provided here to the software project manager, so that they can better control the project's progress.

Let $T(\bullet)$ is a function indicating the token type of places, and let $I(a_i) = \{(P_c, P_j, R_k) \mid j = 1, 2,.., k = 1, 2,...\}$, where P_c represents the change specification report; P_j , R_k represent the respective input products and resources of activity a_i. Two kinds of first priorities for the firing of transition T_{si} are defined here as follows :

(a). As $T(a_i)$ = *complete* initially, then

i). $\exists$ $(T(P_j) = discarded$ *and* $S(P_j) = "*") \rightarrow T(a_i) = discarded.$
ii). $\exists$ $(T(P_c) = active$ *and* $S(P_j) = "*") \rightarrow T(a_i) = discarded.$

That means no matter which state the token in each of the others $I(a_i)$ is in, the transition T_{si} will be fired to change the token state in a_i from "complete" to "discarded" state. Such that activity a_i will be reactivated again, and a new version of product P_j that $P_j \in C(a_i)$ will be created again.

(b). As $T(a_i) = active$ initially, then

i). $\exists$ $((none\ of\ token\ in\ P_j\ or\ T(P_j) \neq active)$ *and* $S(P_j) = "*") \rightarrow T(a_i) = suspended,$
ii). $\exists$ $((none\ of\ token\ in\ R_k\ or\ T(R_k) = empty)$ *and* $S(R_k) = "*") \rightarrow T(a_i) = suspended.$

That means no matter which state the token in each of the others $I(a_i)$ is in, the transition T_{si} will be fired to change the token state in a_i from "active" to "suspended" state.

5. Implications of PM-Net in Software Project Control

The PM-Net model can analyze a project's properties for the sake of determining if activities and subactivities are connected, and if the project plan is complete, consistent and well-executed. Moreover, PM-Net has modified some of DesignNet's functions and extended its advantages in the following ways :

(1). The necessary condition for enabling and firing a transition in DesignNet is based on whether all of its input places are active. For a software project, some activities may have begun before all of its input places become active. For example, when the input resource of an activity is active, the project personnel may start to work on that activity before other input places become active. Therefore, if the activity is restricted so that it cannot be activated until each of its input places has at least one active token in it, the actual status of software development progress will not be represented, especially when an activity has to be activated for the purpose of readjusting the schedule. The "control state operator" is therefore put here onto the input *arc* of transition T_s so as to modify the transition T_s firing rule of DesignNet. A project manager can control the state of control operator for enabling a transition to put an activity on "active" or "suspended" status. The manager can even change the operator from "*"(*AND*) to "+"(*OR*) for the purpose of activating an activity, then change the operator back from "+" to "*", depending on the judgement of the manager;

(2). PM-Net utilizes the DFD technique for decomposing the activities into their bottom- level. The model not only builds up the relationship of an activities products and resources from the bottom-level, it can also display the relation of activities, products and resources of any higher level. All of the rules which fire a transition, create a token and activate an activity are applied to the bottom-level of PM-Net. Information regarding higher levels e.g. project current progress status is propagated from the bottom-level. With PM-Net, a manager can inquire into the information related to project progress from the top-level down to the bottom-level necessary so as to obtain the information required. A top-level manager, for example, may only need to review the overall project's progress, but a bottom-level manager needs to know the executing status of a task. PM-Net provides a flexible representative method for meeting various kinds of requirements for different levels of management. This is the greatest advantage of PM-Net;

(3). Changing the requirements after the requirement specifications have been completed is often requested by the user owing to the nature of a software project. How to ascertain what activities will be affected by a specification change becomes an important issue. In DesignNet, one would start by identifying the token associated with that change and then the project personnel would trace it through the design, implementation and testing activities it triggers, and finally locate the resulting modules. In PM-Net, all one needs to do is to determine what input place in the lowest level for the change of requirements will affect and then the model will automatically trace through the activities and products that could be affected by that change. It will change the place token types for the activities or products that are affected into "discarded" or "suspended" states. The traceability of PM-Net is more powerful than that of the DesignNet;

(4). An activity at a higher-level may have one or more output products. DesignNet cannot represent information involving what output products have been completed and what is still to be created or has not yet started until all of its output products are completed. PM-Net will more precisely represent the project status to a higher level manager. Therefore, for an activity being executed at a higher level, a manager can be told what output has been completed, what is still created, and what has not yet started being created.

6. Points of Clarification

(1). PM-Net model is an open system in which some tokens are provided at certain places by project personnel who must be able to determine whether an activity has succeeded or whether backtracking to previous activities is necessary. The constraints as compared with DesignNet are resultant because this model depends more on project personnel support in that personnel must also determine when and where a control state operator type in the lowest level should be changed.

(2). PM-Net functions by firing transition rules at the bottom-level. The higher-level graphs do not follow the transition firing rules. The token state at any place in PM-Net is reflected by its token state at the lower-level dependent places. Project personnel cannot change any token state in the higher-level graphs of PM-Net. A PM-Net at the higher-level is only a display graph that represents integrated and summarized information regarding software project progress for the sake of more efficiently assisting higher-level managers to understand and control the project.

Examples

Some examples are next provided here for illustrating the functions of PM-Net when applied to software project control. The first level activities of a software project are partially illustrated in *Fig. 4*. No status operator or failure report are presented in *Fig. 4* since a_1 and a_2 are not the bottom-level activities. *Fig. 5* illustrates the second level activities which is decomposed from *Fig. 4*. The activity a_1 is decomposed into a_{11}, a_{12} and a_{13} , and activity a_2 is decomposed into a_{21}, a_{22} and a_{23}. All of a_{11}, a_{12}, a_{21}, a_{22} and a_{23} are bottom-level activities, so there is a control state operator, status operator and failure report in each of the bottom level activities.

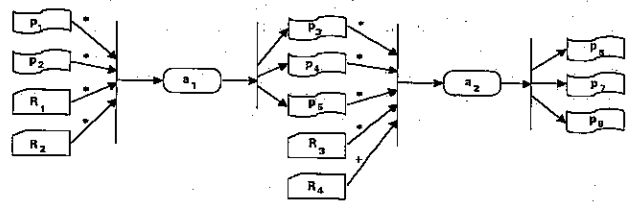

Fig. 4 Part of first-level activities of a software project

The transition T_{s1} is observed in *Fig. 5* to be fired and will create an active token in activity a_{11} if each of its input place P_2 and R_{11} have one active token in it. When a_{11} has an active token, it means a_{11} is being executed, then the transition T_{f1} will be fired and create a "creative" token in its output place P_3. The project personnel can change the control state operator from "+" to "*" in arc P_1 and R_{21} whenever it is necessary. Following the control state operator has been changed to "*", neither one of P_1, or R_{21} has an active token in it; the transition T_{s1} will then be disabled, and the token in a_{11} will be changed from "active" to "suspended" state. Meanwhile, T_{f1} will be fired and the token in place P_3 will also be changed from "creating" to "suspended" state. According to the rules defined here in the previous section, the token

state in a_1 has to be propagated from its subactivities, which means a_1 will also have an active token in it when the token in a_{11} is active. If a_{11} have been successfully completed, the project staff would then assign an active

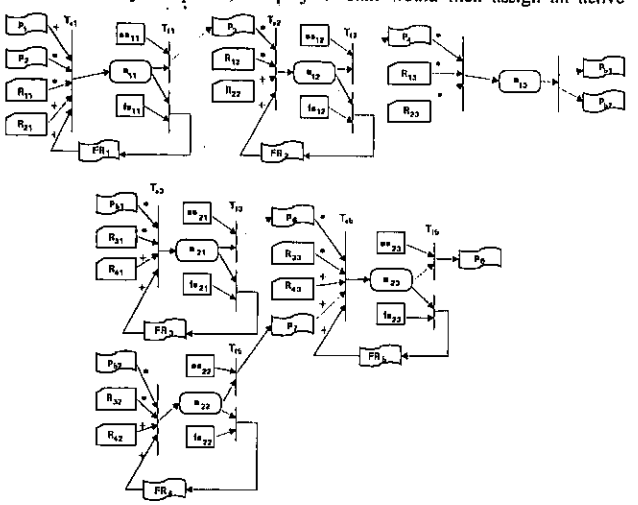

Fig. 5 Second-level activities that decompose from Fig. 4

token to the *success status operator*, and the transition T_{f1} will be fired and the token in place P_3 is changed from "creating" to "active" state. Meanwhile the token in a_{11} will be changed from "active" to "complete" state, and the token in place R_{11} will be changed from "active" to "discarded" state. If a_{11} fails to be completed, the project staff then have to assign an active token to the *fail status operator*, and FR_1 will create an active token. The respective tokens in a_{11} and R_{11} will be changed from "active" to "failure to complete" and "discarded" state, and the *control state operator* in arc FR_1 will be changed from "+" to "*" state. When the token in a_{11} is changed to "failure to complete" state, the transition T_{f1} will be fired again, and the token in place P_3 will be changed from "creating" to "suspended" state.

As a new active token in resource place R_{11} is created, to the number of marks μ in R_{11} will be added 1, and the transition will be enabled and fired again, then the token in a_{11} will be changed from "failure to complete" to " discarded" state, and a new version of an active token will be created instantly. When a new active token is created in a_{11}, to the number of marks μ in a_{11} will be added 1, and the transition T_{f1} will be enabled and fired , then the token will be changed from "suspended" to "creating" state again. If place P_3 and R_{12} are "active", the transition T_{s2} will be fired and will create an "active" token in a_{12}. Then T_{f2} also will be fired and will create a "creating" token in P_4. When the token in a_{11} is "complete", in a_{12} is "active", then the token in the first level of activity a_1 will be in "active" state, and its output product P_3, P_4 will be in "active" and "creating" states respectively.

Fig. 6 is another example which illustrates the variations of a project's activity progress status in PM-Net. At the beginning, activities a_1 and a_{11} at its input places are observed in *Fig. 6a* to have no tokens in them.

Each of places P_1 and R_1 are observed in *Fig. 6b* to have one active token, the transition T_s is enabled, which creates an "active" token in a_1, then transition T_{f1} is enabled and creates an "creating" token in place P_4. In *Fig. 6c*, because both of P_2 and P_3 have active tokens in them, so transition T_s is disabled and the active token in a_1 is changed to "suspended" state, then transition T_{f1} is also enabled and changes the token in P_4 from "creating" to "suspended" state. In *Fig. 6d*, both products P_2 and P_3 have one active token in them, so transition T_s will be enabled and re-activate the token in a_1. Once the token in a_1 is active, T_{f1} will be fired and token in P_4 will be changed from "suspended" to "creating" state. In *Fig. 6e*, if one assigns an active token in fail status operator fs_1, then transition T_{f2} will be fired and an active token in failure report FR_1 will be created. Once FR_1 is in "active" state, the token in a_1 will be changed to "failure to complete" state, then the token in P_4 will be changed to "suspended" state and the token in R_1 and fs_1 will be changed to "discarded" state, as represented in *Fig. 6f*. In *Fig. 6g*, when a new token is

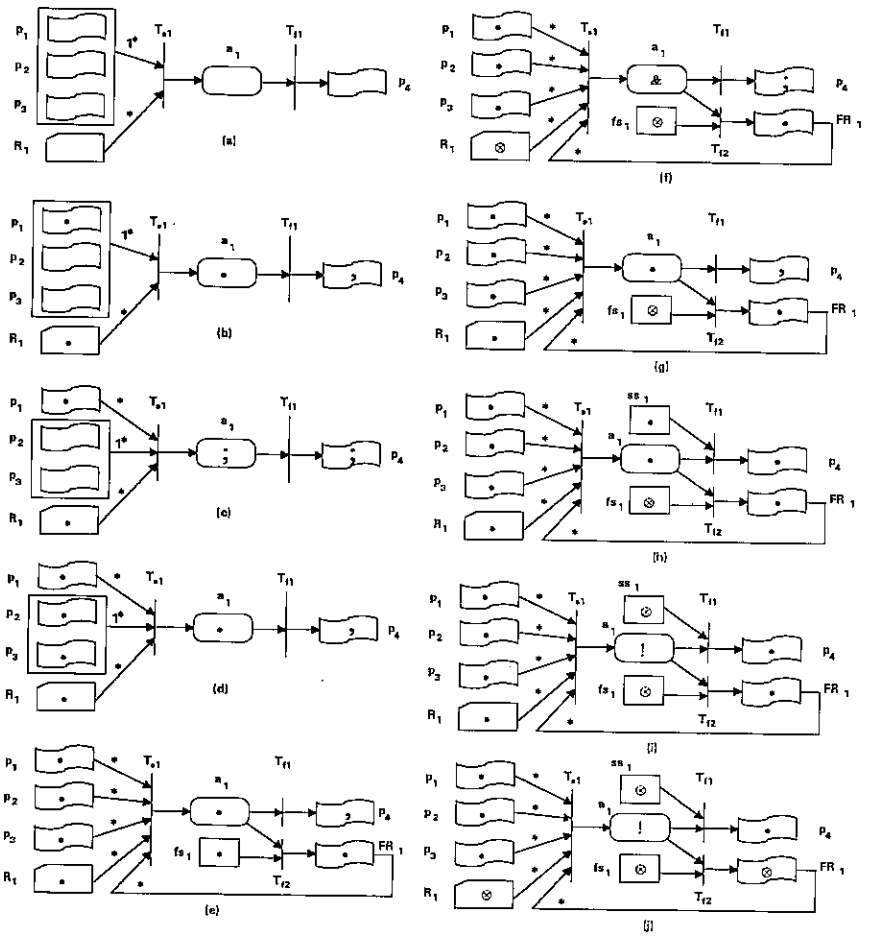

Fig. 6 An example representing the variation of token states in PM-Net

created in R_1, the transition T_s will be fired again and will change the token in a_1 to "discarded" state first, then create a new active token in a_1 again. As a_1 is active, the transition T_{f1} will be fired to change the token in P_4 to "creating" state, as represented in *Fig. 6h*. *Fig. 6i* and *Fig. 6j* represent the variation of token type in some places when an activity is completed.

An example that represents the effect of a requirement specification change on a project's progress is provided in *Fig. 7*. Only activity a_5 is observed from *Fig. 7a* to be executed, all of the others are completed. However, once someone requests to change the requirement specifications, the project personnel will assign a requirement change report for the appropriate site in PM-Net. Once the project staff has implemented the requirement change report P_c at the input place of activity a_2, the activity a_2, a_3, a_4, a_5, and product P_4, P_5, P_6, P_7 will be affected, and the tokens in all of the affected activities will be changed from "complete" or "active" to "discarded" or "suspended" state, the token in all of the affected products will be changed from "active" or "creating" state to "discarded" or "suspended" state, as represented in *Fig. 7b*. *Fig. 7c* represents a new "active" token created in resource place R_2, which will then cause the respective tokens in a_2 and P_4 to be changed from "discarded" to "active" and "creating" state.

regarding whether to reallocate software development resources or to adjust the development schedule in the face of problems of schedule delay or cost overrun. PM-Net can basically integrate a vast amount of well organized information for software development control, but it must remain dependent on a data base system. The concept of repository in CASE tools that attempts to place all useful information into a common data base could be mapped onto the development of a complete environment for software project control system design.

Large software development projects can take many forms and exhibit hybrid patterns of behavior. These development projects must be able to handle all of the possibilities that may occur so as to make such a control system complete. PM-Net therefore seems to be an appropriate model that could be utilized to develop a software project management tool. A framework is currently being developed in our research for software project management with PM-Net as the key feature. The framework will hopefully also become an essential starting point in the development of a fully appropriate system for software project control.

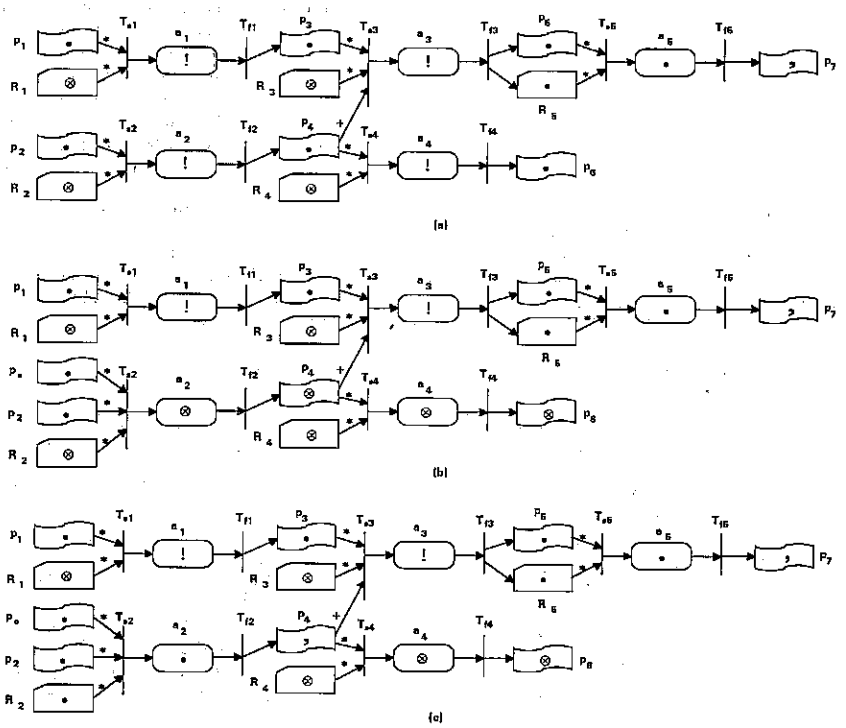

Fig. 7 An example representing the effect of requirement changes in PM-Net

Discussion

Far too little effort has been spent in consideration of how to aid the managers of a software development process. This is surprising, as so much of what makes large-scale software development distinctive is the large managerial component involved [14]. A software project management model for representing and controlling the rate of progress of an software project in execution has, however, been presented here. PM-Net inherits the power of DesignNet. It can describe and monitor the software development process. Tokens are objects with specific properties. Transition firing is a nonvolatile process and creates new token instances providing information relevant to scheduling. Whenever transitions are fired, the project execution history is recorded by the instances created. PM-Net also enhances the power of DesignNet and makes it more suitable for representing and controlling the project's progress. PM-Net utilizes DFD analysis technique for the sake of decomposing activities from top-level to bottom-level. Therefore, PM-Net can represent the project's progress and provide different levels of detailed information for the requirements of different kinds of managers. The modifications that relate to the transition firing function, token types classification, and "control state operators" that put onto the transition T_s input arc further eliminate the inadequacies of DesignNet., The manager can more accurately control software development progress on the basis of this modification. PM-Net will help the manager to make the right decision

References

1. Abdel-Hamid, T. K., and Madnick, S. E., "Impact of Schedule Estimation on Software Project Behavior," *IEEE Software*, May 1986 .
2. Abdel-Hamid, T. k., "On the Utility of Historical Project Statics for Cost & Schedule Estimation," *Journal of System & Software*, 1990.
3. Abdel-Hamid, T. K., and Madnick, S. E., *Software Project Dynamic - An Integrated Approach*. Englewood Cliff, NJ: Prentice-Hall Inc, 1991.
4. Agresti, W. W., "What are the New Paradigms," in *New Paradigms for Software Development : Tutorial*. Washington, DC : IEEE Computer Society, pp. 6 - 10, 1986.
5. Baer, J., Bovet, D., and Estrin, G., "Legality and Other Properties of Graph Models of Computations," *Journal of the ACM*, Vol.17, No.3, pp.543 - 554, July, 1970.
6. Boehm, B. W., *Software Engineering Economics*. Englewood Cliff, NJ : Prentice - Hall Inc, 1981.
7. *Business System Planning, Information Systems Planning Guide*, 4th edition, IBM Corp., N.Y., 1984
8. Cerf, V., "Multiprocessors, Semaphores, and a Graph Model of Computation," *Ph.D. Dissertation*, Computer Science Department, University of California, Los Angeles, California, April, 1972.

9. Curtis, B., Krasner, H., Shen, V., and Iscoe, N., "On Building Software Process Models Under the lamppost," in *Proc. 9th Int. Conf. Software Engineering* Monterey, CA, pp.96 - 103, Mar. 1987.
10. Davis, C. G. Edition, *Project Management : Techniques, Applications, and Management Issues* : Industrial Engineering and Management Press, 1983.
11. Dowson, M., "Iteration in Software Process," in *Proceedings 9th Int. Conferences, Software Engineering*, Monterey, CA, pp.36 - 39, Mar. 1987.
12. Horowitz, E., and Sahni, S., *Fundamentals of Data Structure in Pascal, Rockville*, MD : Computer Science Press, 1985.
13. Kerzner, H., *Project Management, A Systems Approach to Planning, Schedule, and Controlling*. New York : Van Nostrand Reinhold, 1984
14. Liu, L. C., and Horowitz, E, "A Formal Model for Software Project Management," *IEEE Transactions on Software Engineering* Vol.15 No.10, pp.1280 - 1293, Oct., 1989.
15. Londeix, B., *Cost Estimation for Software Development*, Reading , MA : Addison-Wesley Publishing Co, 1987.
16. Macro, T. D., *Structure Analysis and System Specification*. New York, NY : Yourdon Inc., 1978.
17. Peterson, J. L., "Petri Nets," *ACM Computer Surveys* Vol.9, No.3, pp.223 - 252, Sep. 1977.
18. Peterson, J. L., *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ : Prentice - Hall, Inc, 1981.
19. Royce, W. W., "Managing the Development of Large Software Systems," In *Proc. 9th Int. Conference, Software Engineering*, Monterey, CA, pp.328 - 338, Mar. 1987.
20. Wiest, J. D. and Levy, F. K., *A Management Guide to PERT/CPM Prentice* - Hall, Englewood cliffs, NJ, 1977.
21. Winston, P.H., *Artificial Intelligence 2nd. edition*, Reading, MA : Addison-Wesley Publishing Inc., 1984.
22. Yau, S. S., and Caglagan, M. U., "Distributed Software System Design Representation Using Modified Petri Nets," *IEEE Transactions on Software Engineering* Vol. SE - 9, No. 6, pp.733 - 745, Nov. 1983.
23. Yiannis, E. P., and Thomas L. C., "Specification and Analysis of Parallel/Distributed Software and Systems by Petri Nets With Transition Enabling Function," *IEEE Transactions on Software Engineering*, Vol. 18, No.3, pp.252 - 261, Mar. 1992.
24. Yourdon, E., and Constantin, L. L., *Structured Design*, New York : Yourdon Inc, 1975.