An Approach to Software Process Management Based on Formal Process Modeling and Analysis*

Sang-Yoon Min, Ik-Joo Han, Wei-Jin Park, and Doo-Hwan Bae
Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1 Kusong-dong, Yusong-gu, Taejon 305-701, Korea
E-mail : {sang, ijhan, wjpark, bae}@salmosa..kaist.ac.kr

Abstract

In process-driven software development environment, process management plays an important role during the process execution. In order to achieve effective management of software processes, managerial decision makings on development process need to be based on the analysis of the software processes regarding the development environment. In this paper we propose a new approach to process management through the formal process modeling and analysis techniques, called MAM nets based on Pr/T net formalism. For the effective modeling and analysis, MAM nets provide high-level modeling constructs for both process activities and human resources. MAM nets not only support the standard Petri-net analysis techniques but also provide additional analysis techniques as well as the process enactment mechanism for the effective management of software processes.

1. Introduction

With the increasing interest in improving the effective* ness and predictability of software development activities, software process management became one of the most important issues in the software engineering community. Specially, in the process-driven software development environment, process management techniques play decisive roles in carrying out the software projects.

For the effective management of software development processes, not only well-designed software process model is necessary, but also the process model itself needs to be analyzed and managed precisely so that the management activities can be performed based on reasonable and rational decision makings.

Growing interest in developing effective approaches to software process management has led to focus on the employing formalisms on software process modeling. Employing formalisms provides the enhanced capabilities such as fine-grained modeling and formal analysis. In recent years, various formal approaches to software process modeling have been introduced, including Petri-net based approaches.

In order to make reasonable and rational decisions during the process management, some means for systematic analysis of process execution and related environment factors are necessary. However, most of the formal approaches mainly are focused on process modeling techniques rather than the techniques for the process management. Their formal analysis techniques are primarily aiming to designing mathematically correct model. The formal modeling techniques provide fundamental means for increasing the maturity of software development with correct software process model. However, merely establishing good software process model does not necessarily guarantee the effectiveness of actual process activities. For the effective process management upon a software process model, the managerial activities should be guided based on the analysis of software processes in terms of their execution.

For this purpose, we propose new techniques for software process management using a high-level Petri-net based approach to software process modeling and analysis , namely MAM nets(Modeling, Analysis, and Management nets). MAM nets provide the capability of analyzing software processes in terms of their execution in order to aid effective decision makings for managerial activities, as

* This work was partially supported by the Ministry of Information and Communication Korea, and Intervac Co. under industrial-educational joint project, "Software Project Management System Based on Development Process Analysis."

0-8186-8271-X/97 $10.00 © 1997 IEEE

well as model verification. In addition, MAM nets provide the high-level modeling constructs for the efficiency and easiness of modeling software processes. The next section describes the background of software process modeling and management. In section 3, the formal definitions, graphical notations, and modeling mechanisms of MAM nets are described. In section 4, we explain the supported analysis techniques for software process management as well as model verification. The process enactment mechanism of our approach is given in section 5. Finally, the conclusions and the future works are given in section 6.

2. Software process modeling and management

Considering the current trend toward process-driven software development, software process modeling and process management have an inseparable relation to each other. The growing scales of the software development projects and organizational structures of today reveal the limits of the traditional approaches to process modeling and management.

In the traditional approaches, not only the modeling was coarse grained and informal, but also the process management were mostly led by abstract guidelines rather than systematic analysis of the software process model. The abstraction and informality of the traditional approaches produce the fundamental discrepancy between the software process model and the process management.

For the systematic conducting of process-driven software development, the discrepancy need to be eliminated. This means the process management strategy must be based on the precise analysis of the software processes. To fulfill the needs for the capabilities of analyzing software processes, recent researches on software processes have increasing attention to formal process modeling, which support various analysis techniques.

2.1 Formal Approaches to Software Process Modeling

The significant advantages of formal process modeling are its capability of formal analysis and fine-grained modeling. Of course the capabilities supported by each approach are varied with the kinds of the base formalism. The formal approaches to software process modeling can be classified according to their base formalisms such as process programming, object-oriented modeling, AI, and Petri nets. Surveys and comparisons of the various approaches can be found in [ABGM92] [CKO92][Kling92].

Among those approaches, our interest focuses on the Petri-net based approaches to software process modeling. There have been introduced several approaches based on Petri-net formalism. FUNSOFT nets[EG95] and SLANG [BF93][BFGL94] are two representative techniques from Petri-net based approaches. FUNSOFT nets are based on high-level Pr/T nets. FUNSOFT nets provide various kinds of modeling constructs for places, arcs and transitions to support dense representations of software processes. SLANG is built on the top of a high-level Petri-net formalism, called E/R nets[BFGM91] [GMM91]. SLANG possesses a distributed notion of state described by the marking of the net.

2.2 Software processes management and process analysis

As we mentioned earlier, it is necessary that software process management should be based on the systematic analysis of software processes, in order to aid effective process-driven software development. To support the management activity through the analysis of software processes, the analysis mechanism of software processes should provide the capability of analyzing the behavior and the efficiency of the process execution under a certain environment setting as well as the capability of verifying correctness of the model.

In order to perform the analysis of the behavior and the efficiency of the process execution, the elements of the development environment should be accounted. In other words, the elements which characterizing the development environment should be integrated into the software process model to take advantage of the analysis formalisms supported by the formal modeling techniques.

Although the existing Petri-net based approaches support various powerful mechanisms for modeling and analysis of software processes, they do not provide the explicit modeling constructs and adequate analysis mechanisms for software processes and elements of development environment, specially the human resources.

Even if the management decisions were made according to the analysis of the process model and environmental settings, the decisions could not be guaranteed as the best solution due to the complex dynamic nature of software processes. Indeed, the actual software processes are heavily human-oriented, and can not be fully formalized[AM91]. Rather, it would be more desirable to test manager's multiple decisions using systematic analysis and simulation, and to make a choice among the several alternatives.

For these reasons, we have developed a Petri-net based approach to software process modeling and management to support effective process-driven software development. The detailed explanations of our approach are given in the next section.

3. A Petri-net based approach to software process modeling and management

In this section, we describe our approach to software process modeling and management, called MAM nets. Upon the modeling and analysis formalism, we have also developed various additional analysis techniques to support process management. First, the underlying ideas and the basic concepts of MAM nets are introduced briefly. Following the introduction, the formal definitions and graphical notations are described.

3.1 Introduction to MAM nets

The base formalism of MAM nets is derived from Pr/T nets. MAM nets provide two types of the main modeling constructs: *ProcessStep* and *AgentAllocationUnit*, for the efficiency of modeling mechanism as well as the integrated modeling of software processes and the elements of development environments, specifically the human resources.

A ProcessStep corresponds to a process activity in the actual software development. AgentAllocationUnit takes a role of an agent pool that allocates the assigned process agents to each ProcessStep. In MAM nets, software processes are modeled as a set of ProcessSteps and a single AgentAllocationUnit, the pool of human resource.

For effective process management, MAM nets also provide various kinds of analysis mechanisms for process management as well as process enactment. The formal definition of MAM nets is given in the following section.

3.2 Definition of MAM nets

A MAM net is defined as a tuple (*PS, AU, IP, OP, PI, AI, T, F, P; A, M0*). Each element of the tuple is defined as following.

PS = {ps1, ps2, ..., psn} defines a finite set of *ProcessSteps*. A ProcessStep represents a process activity in software processes. *AU = AgentAllocationUnit* defines a pool of available process agents. A ProcessStep is supposed to send a request to AgentAllocationUnit in order to receive the assigned process agents before its execution. Then, AgentAllocationUnit allocates requested process agents to the requesting ProcessStep if they are available. After the completion of the execution, the ProcessStep restores the process agents to AgentAllocationUnit so that they should be available to another request. *IP = { ip1, ip2, ..., ipn}* denotes a set of artifact-input ports of ProcessSteps. *OP = {op1,op2,...opn}* denotes a finite set of artifact-output ports of ProcessSteps. *PI* represents a set of process agent allocation ports of ProcessSteps. In a MAM net, all ProcessSteps must communicate with AgentAllocationUnit for process agent allocation. PI is a set of the communication ports of ProcessSteps for this purpose. Each ProcessStep has three communication ports. One of them is for sending requests for process agents, another is for receiving allocated process agents from AgentAllocationUnit, and the other is for releasing the finished agents. Each of these ports is called *pq*(ProcessStep's agent-reQuesting port), *pc* (ProcessStep's agent-reCeiving port), and *pl*(ProcessStep's agent reLeasing port), respectively. Thus, PI can be defined as follows :

$PI = (PQ \cup PC \cup PL)$, where
$PQ = \{pq_1, pq_2, ..., pq_n\}$ is a finite set of agent-requesting ports,
$PC = \{pc_1, pc_2, ..., pc_n\}$ is a finite set of agent-receiving ports,
$PL = \{pl_1, pl_2, ..., pl_n\}$ is a finite set of agent-releasing ports, and
PQ, PC and *PL* are exclusively disjoint.

AI denotes a set of process agent allocation ports of AgentAllocationUnit. AgentAllocationUnit also has three kinds of communication ports corresponding to *PI*. We named these three ports as *aq*, *as*, and *ar*. Port *aq* is request-receiving port that receives agent request from pq_i. Port *as* is the port for sending requested agents. Port *ar* is agent-receiving port that receives the released agents from pl_i of ProcessSteps. Thus AI is defined as the set, *{ aq, as, ar }*. *T = {t1,t2,...tm}* is a finite set of transitions. *F* = a finite set of arcs. For clarity, we divide *F* into four disjoint subsets, *F1, F2, F3* and *F4*. *F1* is the set of arcs connecting two sequential ProcessSteps or forming rework cycles. *F2* is involved in the situation of ProcessStep abstraction. In case of ProcessStep abstraction, the input-artifacts of the abstracted ProcessStep must be distributed to the sub-ProcessSteps that are contained in the abstracted ProcessStep. Also, the final output-artifacts of sub-ProcessSteps must be merged into the artifacts-output port of the higher level ProcessStep. The arcs in *F3* are also involved in the situation of ProcessStep abstraction regarding the connections between *PI* of the subProcessSteps and *PI* of the higher level ProcessStep. The arcs in *F4* represent the connections between AgentAllocationUnit and ProcessSteps regarding agent request, allocation, and restoring . Thus the set of arcs, *F* is defined as follows :

$F \subseteq (F1 \cup F2 \cup F3 \cup F4)$, where
$F1 \subseteq (OP \times T) \cup (T \times IP) \cup (P \times T) \cup (T \times P)$
$F2 \subseteq (IP \times T) \cup (T \times IPsub) \cup (OPsub \times T) \cup (T \times OP)$
$F3 \subseteq (PQsub \times T) \cup (T \times PQ) \cup (PC \times T) \cup (T \times PCsub) \cup (PLsub \times T) \cup (T \times PL)$, and
$F4 \subseteq (PQ \times T) \cup (T \times aq) \cup (as \times T) \cup (T \times PC) \cup (PL \times T) \cup (T \times ar)$

Set P denotes a set of places. Set A represents a set of annotations. There are four kinds of annotations. They are An, As, At, and Af. An stands for a set of general annotations in a MAM net. Each As, At, and Af stands for set of annotations for ports, transitions, and arcs, respectively. Thus, the set of annotation A is $(An \cup As \cup At \cup Af)$. M_0 denotes initial marking.

Based on above formal modeling constructs, MAM net provides consistent ways to model various types of software processes with dense descriptions of process activities, process artifacts, precedence of processes, process agents and their allocations. Also, these formal definitions of MAM net are used as the theoretical bases for transforming MAM net into Pr/T nets in order to apply standard Pr/T net analysis techniques.

3.3 Graphical notations and modeling mechanism

In this section, we describe the graphical notations for the modeling constructs of MAM nets. As shown in Figure 1(a), a ProcessStep is depicted as a rounded rectangle with the five ellipses on its edges. The ellipse on the left of the rectangle, denoted by *ip*, represents the artifacts-input port of ProcessStep. Similarly, the ellipse on the right of the rectangle, denoted by *op*, is the artifacts-output port. The three ellipses on the bottom of the rectangle, denoted by *pq*, *pc* and *pl*, represent the agent-requesting port, the agent-receiving port, and the agent-releasing port, respectively. As depicted in Figure 1(b), AgentAllocationUnit is represented as a rounded rectangle with three ellipses on the top. The three ellipses, denoted by *aq*, *as*, and *ar*, are the request-receiving port, the agent-sending port, and the agent-receiving port, respectively.

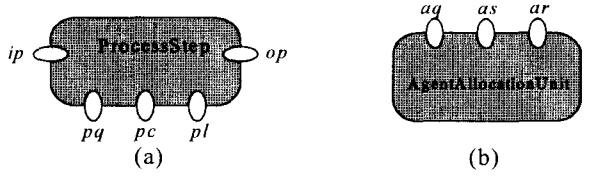


Figure 1 Graphical Notations of ProcessStep and AgentAllocationUnit

Figure 2 illustrates typical examples of basic modeling mechanisms of MAM nets. As described in the definitions, the internal connections of hierarchically abstracted ProcessStep can be modeled consistently regarding artifact distribution, merging, and agent allocation as shown in Figure 2(a). One of the most important modeling mechanism is the connections between ProcessSteps and AgentAllocationUnit. The connections are very straight forward and trivial as shown in Figure 2(b).

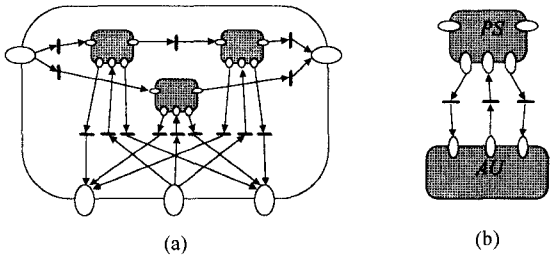
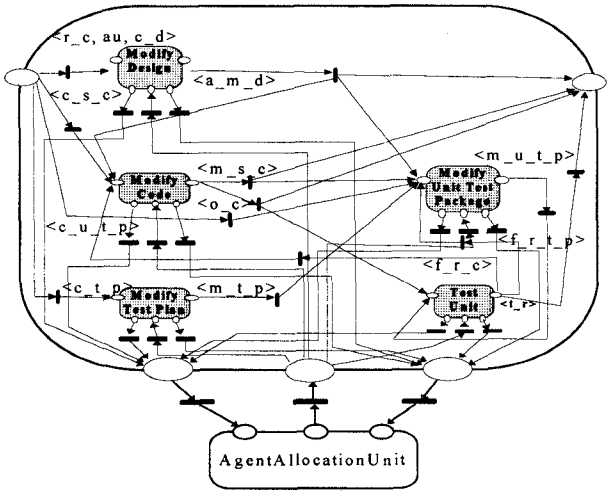


Figure 2 Examples of Basic Modeling Mechanism : (a) a hierarchical abstraction of ProcessSteps, and (b) connection between a ProcessStep and AgentAllocationUnit

In order to present our experimental study, we modeled ISPW-6 software process example[KEF+90] with MAM nets. The modeled high-level process activities are shown in Figure 3.



<r_c>:requirements change <au>:authorization
<m_t_p>:modified test plan <a_m_d>:approved modified design
<c_d>: current design <c_s_c>: current source code
<f_r_c>: feedback regarding code <u_t_p> : unit test package
<m_s_c>:modified source code <c_t_p>: current test plan
<f_r_t_p> : feedback regarding test package <t_r>: test results
<m_u_t_p>: modified unit test package <o_c> : object code

Figure 3 ISPW-6 software process example modeled in MAM nets

The five ProcessSteps shown in Figure 3 are the high-level process activities corresponding to *modify design*, *modify code*, *modify test plan*, *modify unit test package*, and *test unit* as described in [KEF+90]. Each of these ProcessSteps can be modeled into lower level ProcessSteps according to the original description. For example, the

internal sub-ProcessSteps of modify design ,consisting of two sub-ProcessSteps, *change design* and *review design*, can be modeled as shown in Figure 4.

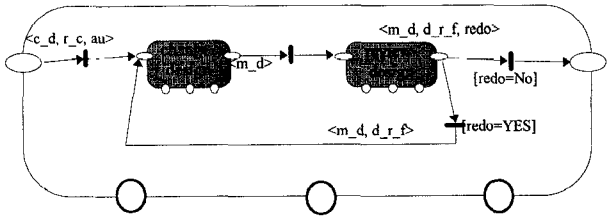


Figure 4 ProcessStep: Modify Design

4. Analysis for process modeling and management

In this section, we describe the supported formal analysis techniques of MAM nets. Our approach toward the analysis is driven by two perspectives, analysis of model structure used in process modeling phase, and analysis to aid managerial decision makings in the process management phase, as described in Figure 5. The former mainly concerns the verification of correctness of the process model, and the later focuses on the analysis of the process execution regarding the human resource constraints and the process activities upon the process model.

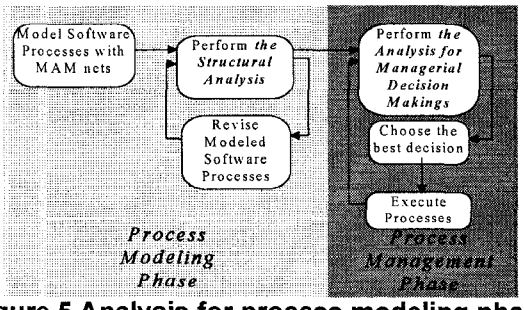


Figure 5 Analysis for process modeling phase and process management phase

4.1 Analysis of model structure

For the analysis of structural correctness of process model, the standard Petri-net analysis techniques such as deadlock detection and trap detection, are used. To apply the existing standard Petri-net analysis techniques, the modeled software processes need to be transformed into a type of an existing Petri-net class. In our approach, we transform MAM nets into equivalent Pr/T nets through the unfolding mechanism. The unfolding of MAM nets is accomplished by two transformation procedures. The first procedure is to transform the atomic(i.e. not abstracted) ProcessSteps and the AgentAllocationUnit into corresponding Pr/T nets. The second procedure is to transform the other overall net structures into Pr/T nets.

4.1.1 Unfolding atomic ProcessStep and AgentAllocationUnit.

As described in section 3, the internal structure of an atomic ProcessStep is defined in terms of Pr/T nets as shown in Figure 6. All the ports(i.e. the ellipses on the edges) are basically places. The transition on the left fires when input artifacts arrive at port *ip*. This firing results in placing a token representing an agent request in port *pq* while transferring the input artifacts to the temporal place in the middle. When requested process agents arrive at port *pc* from AgentAllocationUnit, the activity transition is enabled and fires. The firing of the activity transition produces output artifacts into port *op*, and releases the finished agents into port *pl*. As we can easily recognize, the transition on the right is the transition corresponding to the execution of actual process activity. <i_a_n> and <o_a_n> refer the names of input and output artifacts, respectively, such as <source_code> and <object_code>. <a_c> is the code standing for assigned process agents for the ProcessStep. <pid> is unique identifier for each ProcessStep.

The internal structure of AgentAllocationUnit is depicted in Figure 7. When agent request arrives at port *aq*, AgentAllocationUnit first checks the request with current available agents. If available, the requested agents are placed in port *as*, and the current available agents are updated. If not available, the request is sent back to port *aq*, and waits until the requested agents are available. When a ProcessStep releases its agents, the agents arrive at port *ar*. Then, AgentAllocationUnit updates the current available agents according to the restored agents.

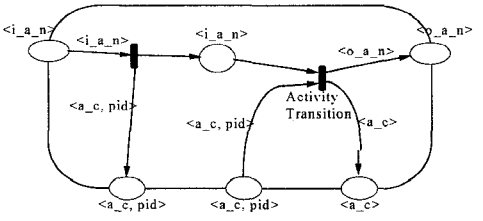


Figure 6 Unfolded atomic ProcessStep

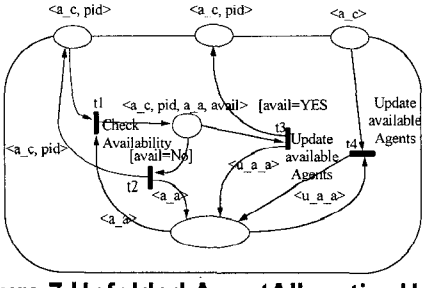


Figure 7 Unfolded AgentAllocationUnit

4.1.2 Unfolding overall net structures

This section describes the mechanism of unfolding overall net structures into the equivalent Pr/T nets. Unfolding the overall net structures of MAM-Nets means unfolding the high-level modeling mechanisms of MAM nets. The high-level modeling mechanisms of MAM-Nets are the high-level representation of places(ports), arcs, and transitions. Places(i.e. ports) of MAM-Nets can hold more than one type of tokens. Likewise, the arcs and transitions of MAM-Nets can transfer more than one type of tokens at the same time. Using the systematic unfolding mechanism described in [Mur89] and the reduction mechanism described in[Mur89][Rauh90], any modeled software processes with MAM nets can be transformed into the corresponding Pr/T nets without loosing any properties such as liveness, safeness, and boundedness . The basic mechanism of unfolding is simple. By making separate places and arc for each type of token, the modeled MAM nets can be transformed into equivalent Pr/T nets. Examples of the unfolding are shown in Figure 8(a) and (b).

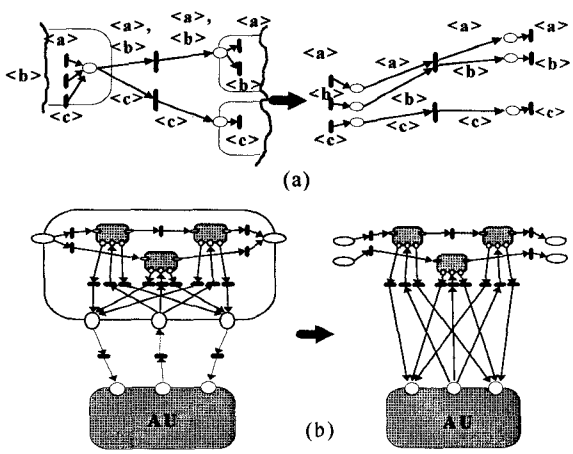


Figure 8 Example of unfolding overall net structures : (a) unfolding the connections involved in artifact transfer and (b) unfolding the connections involved in agent allocation.

4.2 Analysis for managerial decision makings

The analysis to aid managerial decision makings during the process management are driven by the timed Petri-net formalism accounting the relationships between the modeled software processes and the environmental properties, specially the process agents. The goal of this analysis is to provide more analytic information to aid correct decision-makings during the activity planning and enactment. Using the analysis techniques, we can examine following properties of the software processes modeled with MAM nets:

- Cumulative time delay of process activity
- Spots of agent conflicts in a given schedule and their influence.
- Concurrency of ProcessSteps at any given time
- Minimum requirements of total manpower to finish the project
- Minimum number of agents to support maximum concurrency of ProcessStep
- Agent utilization (if no conflict occurs)

For the analysis of above properties, we assume that the expected duration on modeled software processes are predictable prior to the actual development. Using the timed Petri-net formalism, we can compute the cumulative time delay of process activity. Upon the timed Petri-net formalism, we have developed additional analysis techniques to extract the other properties stated above.

4.2.1 Timed Petri-net based approach

Timed Petri net is a well-known powerful approach to evaluate system performance and resource scheduling through the analysis with delayed transitions. Based on the timed Petri-net properties [Mag83][Mur89][Zub91], the cumulative time delay(CTD) just after the execution of a transition t_i can be defined as a recursive equation as following :

$$CTD(t_i) = D(t_i) + \text{Max}\ (\ CTD(t) \text{ of each } t \in T_{in}), \quad .....(1)$$

where
$D(t_i)$ = assigned delay at t_i
$P_{in} = {}^\bullet t$
T_{in} = set of all t's such that each $t \in {}^\bullet p$, $\forall\ p \in P_{in}$
$CTD(t_i)$ = cumulative time delay just after execution of transition t_i
${}^\bullet t$ is the set of input places of transition t, and
${}^\bullet p$ is the set of input transitions of place p

The recursive equation (1) implies that the cumulative time delay after the execution of transition t_i is the sum of the time delay of t_i and the latest arrival time among its input token. Using the equation (1), we can compute the cumulative time delays of each ProcessStep of modeled MAM nets. The transitions with durations are the activity transitions inside the each atomic ProcessSteps as described in Figure 6. All of the other transitions are considered to have zero duration time.

4.2.2 Experimental Model

For our experimental study of analysis, we extended the ISPW-6 software process example [KEF+90] into more complex state. The original problem description assumes only a single requirement change is made. For more realistic experimental study, we assumed there are two independent requirement changes (*requirement change A* and *requirement change B*). However, the available human resources are assumed to remain unchanged. The assignments of process agents are made based on the original problem descriptions in [KEF+90]. The expected duration of each process activity is arbitrarily determined. The experimental environment settings are shown in Table 1.

Source of Cause	ProcessSteps	duration	Agents
requirement change A	change design(ps_A1)	10 days	D1
requirement change A	review design(ps_A2)	4 days	D1,Q, E1, E2
requirement change A	edit code(ps_A3)	10 days	D1
requirement change A	compile and error check(ps_A4)	6 days	D1
requirement change A	modify unit test package(ps_A5)	8 days	Q
requirement change A	modify test plan(ps_A6)	14 days	Q
requirement change A	test unit(ps_A7)	4 days	D1,Q
requirement change B	change design(ps_B1)	6 days	D2
requirement change B	review design(ps_B2)	4 days	D2, Q, E1, E2
requirement change B	edit code(ps_B3)	6 days	D2
requirement change B	compile and error check(ps_B4)	2 days	D2
requirement change B	modify unit test package(ps_B5)	4 days	Q
requirement change B	modify test plan(ps_B6)	8 days	Q
requirement change B	test unit(ps_B7)	6 days	D2,Q

Design Engineer : D1, D2, Quality Assurance Engineer : Q, Software Engineer : E1, E1

Table 1 Experimental environment settings

Using the computation mechanism of equation (1), the cumulative time delay of each process activity is simulated. The ideal cumulative time delay without considering the constraints from process agents are shown in Table 2. In case of our example, the ideal duration to finish all of the process activities caused by requirement change A is 42 days and 28 days for requirement change B.

CTDs of requirement change A (unit of CTD : day)

	ps_A1	ps_A2	ps_A3	ps_A4	ps_A5	ps_A6	ps_A7
CTD	10	14	24	30	38	14	42

CTDs of requirement change B (unit of CTD : day)

	ps_B1	ps_B2	ps_B3	ps_B4	ps_B5	ps_B6	ps_B7
CTD	6	10	16	18	22	8	28

Table 2 CTD of Ideal Processes

4.2.3 Analysis Mechanism

For the analysis of the other properties mentioned earlier, we have developed an analysis table called *agent-time table*. An agent-time table is composed of columns representing absolute scales of time and rows representing involving performers, the process agents.

Using the information of the CTDs of ideal processes and the environmental setting, each cell of the agent-time table is filled with the adequate ProcessStep ID's. For each ProcessStep, we pick the row(s) corresponding to the assigned agent(s). Next, we determine the column(s) corresponding to expected ideal duration period of the process activity. Then we fill the cells in the cross-section with its ProcessStep ID. The resulting agent-time table of the extended ISPW-6 example is shown in Table 3.

(letter *xi* in each cell stands for ProcessStep ps_*xi*)

Time / Performer	0-2	2-4	4-6	6-8	8-10	10-12	12-14	14-16	16-18	18-20	20-22
D1	A1	A1	A1	A1	A1	A2	A2	A3	A3	A3	A3
D2	B1	B1	B1	B2	B2	B3	B3	B3	B4		
Q	A6 B6	A6 B6	A6 B6	A6 B2 B6	A6 B2	A2 A6	A2 A6			B5	B5
E1				B2	B2	A2	A2				
E2				B2	B2	A2	A2				

Time / Performer	22-24	24-26	26-28	28-30	30-32	32-34	34-36	36-38	38-40	40-42	42-44
D1	A3	A4	A4	A4					A7	A7	
D2	B7	B7	B7								
Q	B7	B7	B7		A5	A5	A5	A5	A7	A7	
E1											
E2											

Table 3 Agent-Time Table of Ideal Processes

Spots of process agent conflicts and their influence : Agent conflicts can occur in two cases ; when a process activity requests a performer who is not available due to the allocation to another activity, or when two or more process activities request the same performer at the same time. The conflicts can be easily detected through the agent-time table. Any cell containing more than one ProcessStep ID is the spot of process agent conflict. The

ProcessStep IDs contained in the conflict cell imply the process activities involving in the conflict. The row and the column of the conflict cell represent the agent in the conflict and the time of the conflict occurs. In our example, many conflicts on QA engineer are observed.

To resolve the conflicts, the project manager needs to apply a scheduling policy. For example, the agent time table of conflict-free process activities rearranged by SJF(Shortest Job First) policy is shown in Table 4.

(letter *xi* in each cell stands for ProcessStep ps_*xi*)

Time / Performer	0-2	2-4	4-6	6-8	8-10	10-12	12-14	14-16	16-18	18-20	20-22	22-24	24-26
D1	A1	A1	A1	A1	A1		A2	A2	A3	A3	A3	A3	A3
D2	B1	B1	B1		B2	B2	B3	B3	B3	B4			
Q	B6	B6	B6	B6	B2	B2	A2	A2	A6	A6	A6	A6	A6
E1					B2	B2	A2	A2					
E2					B2	B2	A2	A2					
Time / Performer	26-28	28-30	30-32	32-34	34-36	36-38	38-40	40-42	42-44	44-46	46-48	48-50	50-52
D1	A4	A4	A4									A7	A7
D2					B7	B7	B7						
Q	A6	A6	B5	B5	B7	B7	B7	A5	A5	A5	A5	A7	A7
E1													
E2													

Table 4 Agent-Time Table of Rearranged Conflict Free Process Activties

Concurrency of process activities : The degree of concurrency of software processes has important meanings in software development. High degree of concurrency implies high efficiency of software development. Therefore, it is important to preserve the concurrency of the process activities in the original software process model during the actual development. From another aspect, excessively high degree of concurrency can provoke difficulties in managing activities. In the agent-time table, the number of distinct ProcessStep ID's of each column implies the degree of concurrency at the corresponding period of time. The degrees of concurrency along with the absolute time for the original processes and the rescheduled processes are shown in Figure 9.

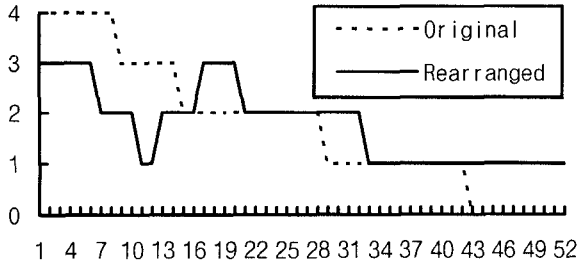


Figure 9 Degrees of Concurrency of Original and Rearranged Processes vs. Time

Minimum requirements of the total manpower to finish the whole processes : The total number of ProcessStep ID's in the agent-time table multiplied by the unit of each column corresponds the minimum manpower needed to finish the all the process activities. In our the example, the whole process activities need 126 man-days of manpower.

Minimum number of performers to support the maximum concurrency of the modeled processes : To support the maximum concurrency through out the development activities, the requested human resources of each process activity must be allocated without delays. The number of ProcessStep ID's contained each column of the agent-time table refers the minimum number of performers needed for the each period of time. The required minimum number of performers at each time of the original and rearranged processes is shown Figure 10.

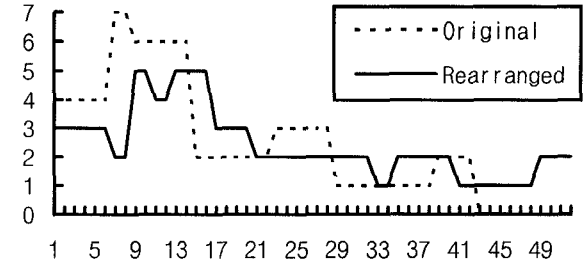


Figure 10 Required Minimum Performers vs. Time

Process Agents utilization : In case that there is no resource conflict, the agent-utilization rate can also be computed through the agent-time table. Blank cells contained in each row imply that the corresponding performer is not being used by any process activity during the period. According to the agent-time table of the rearranged process activities, we get the results as shown in Figure 11.

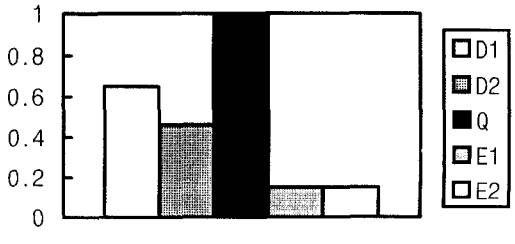


Figure 11 Process Agents (Performer) Utilization Rates

5. Process Enactment

The primary assumption on process enactment is that the software processes have been correctly modeled. To

ensure the efficiency of the process execution, the analysis for the managerial decision makings described in the previous section should be performed whenever any decision needs to be made among several alternatives. After each decision making through the analysis, each actual process activity is enacted according to the process model.

As we explained in section 3, each ProcessStep of MAM nets is designed to request the assigned agents to AgentAllocationUnit when its input-artifacts arrive. Therefore, the tokens representing the agent-request(consists of the ID's of ProcessSteps and the requested agents) sent from ProcessSteps keep arriving at port *aq* of AgentAllocationUnit along with time. Thus, port *aq* of AgentAllocationUnit takes the role of a queue for the ready jobs. Using adequate scheduling policies such as SJF and FCFS, the candidate ProcessSteps for the enactment can be easily identified. The arriving tokens for each unit of time are also corresponding to the ProcessStep ID's contained each column of the scheduled agent-time table.

6. Conclusions and Future Works

In this paper, we described an approach to software process management based on a formal software process modeling and analysis techniques that we have developed. The process modeling, analysis and management approach , namely MAM nets, is based on the formalism of Pr/T nets with time extension. The goals of our approach are to provide the capability to support effective process management through the formal modeling and analysis mechanisms to verify the correctness of modeled software processes, and to aid managerial decision makings. The advantages of our approach could be summarized as below:

- Support for Integrated modeling of software processes and process agents
- Based on a formal modeling method, Pr/T nets
- Support for standard Petri-net analysis techniques.
- Guide the process management through formal analysis techniques such as cumulative time consumption of development activities at any process activity, spots of human resource conflicts and their influence, concurrency of process activities, minimum requirements of total manpower to finish the whole processes, minimum number of performer to support the maximum concurrency of ideal processes, and process agent utilization
- Provides easy and effective mechanism of process enactment

In order to increase the practical applicability of our approach, we need to do further researches in the future. The future works include researching on modeling other environmental properties in addition to human resources and continuing experiments with actual industrial cases.

References

[AM91] T. Abdel-Hamid and S. Madnick. "Software Project Dynamics : An Integrated Approach", Prentice Hall, 1991.

[ABGM92] P. Armenise, S. Bandinelli, C. Ghezzi, and A. Morzenti."Software Processes Representation Languages : Survey and Assessment", *Proceedings of the 4th International Conference on Software Engineering and Knowledge Engineering*, pages 455-462, Capri, June 1992.

[BF93] S. Bandinelli and A. Fuggetta. "Computational Reflection in Software Process Modeling: the SLANG Approach", *Proceedings of the 15th International Conference on Software Engineering*, pages 144-154, May 1993.

[BFGM91] S. Bandinelli, A. Fuggetta, C. Ghezzi, and A. Morzenti. "A Multi-Paradigm Petri Net Based Approach to Process Definition." *Proceedings of the 7th Internatinal Software Process Workshop, Yountville*, October, 1991.

[BFGL94] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. "SPADE : An Environment for Software Process Analysis, Design, and Enactment." *Software Process Modeling and Technology*, pages 223-247, Research Studies Press Ltd., 1994

[CKO92] B. Curtis, M. Kellner, and J. Over. "Process Modeling" *Communications of the ACM, Vol.35, No.9*, pages 75-90, September 1992.

[EG95] W. Emmerich and V. Gruhn. "Software Process Modelling with FUNSOFT Nets" University of Dortment. October 1995.

[GMMP91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. "A Unified High-Level Petri Net Formalism for Time-Critical Systems" *IEEE Transactions on Software Engineering*, Vol.17, No. 2, pages 160-172, February, 1991.

[KFF+90] M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. Rombach. "ISPW-6 Software Process Example" *Preceedings of the 6th ISPW*, page 176-186, October, 1990.

[Kling92] C. Klingler. "A STARS Case Study in Process Definition" *Proceedings of TRW Systems and Software Engineering Symposium*, December 1992.

[Mag83] J. Magott "Performance Evaluation of Concurrent Systems Using Petri Net", *Information Processing Letter 18*, pages 7-13, September, 1983.

[Mur89] T. Murata. "Petri nets : Properties, Analysis, and Applications" *Proceedings of IEEE, Vol.77, No.4*, pages 541-580, April 1989.

[Rauh90] M. Rauhamaa. "A Comparative Study of Methods for Efficient Reachability Analysis". Research Report. Digital System Lab. Helsinki Univ. of Tech. Sept. 1990.

[Zub91] W. Zuberek. "Timed Petri nets : Definitions, Properties, and Applications" *Microelectronics and Reliability*, Vol.31, No.4, pages 627-644, 1991.