SoftPM: a software process management system reconciling formalism with easiness

S.-Y. Min*, H.-D. Lee, D.-H. Bae

Department of Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Kusong-dong, Yusong-gu, Taejon 305-701, South Korea

Received 1 October 1998; received in revised form 17 May 1999; accepted 18 May 1999

Abstract

Various formal approaches to process modeling and analysis have been proposed. With the emerging importance of practicality in this field, easiness in adopting formal technology should be taken into account. In this paper, we propose a PSEE called SoftPM that is based on a high level Petri net formalism for process modeling. To overcome the difficulty in using this formalism, SoftPM exploits a multi-level modeling mechanism for the representation of software processes. SoftPM supports three different levels for process representation. They are cognitive, MAM net, and Pr/T net. Each level has different roles in SoftPM. The cognitive-level modeling provides the means of getting free from difficulties in manipulating the modeling formalism. MAM net takes the role of core modeling formalism in SoftPM. Finally, Pr/T nets support the low-level modeling formalism as an existing Petri-net class. Moreover, SoftPM offers various analysis techniques to aid managerial decision making, as well as conventional Petri-net analysis techniques. Using a Java-based thin client/server architecture, SoftPM smoothly supports execution at distributed heterogeneous platforms over the Internet. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Software process; Process modeling; Process-centered software engineering environment

1. Introduction

Since the last decade, software process has been recognized as one of the major research areas in software engineering. Along with the persistent growth in the scale of software projects, project managers have been burdened with the increasing responsibility of managing development processes. Thus, the software engineering community has been exerting great effort in the field of software process. Various techniques have consequently been proposed and developed. These techniques are widely ranging including process modeling, analysis, execution, assessment and improvement.

Such techniques can be divided into two groups: PSEE (Process-centered Software Engineering Environment) related techniques [1–6] which include process modeling, analysis and execution techniques, and those techniques related to an organizational maturity improvement to which process assessment and improvement techniques belong. In this paper, we concentrate on the techniques of the former group since the latter is mostly concerned with management strategies.

Building a PSEE requires diverse techniques. The foremost core technique to be considered is the process modeling technique, as the direction for building an actual execution environment depends on what kind of a process modeling technique is engaged. In the field of process modeling, several disciplined approaches have been developed with various base paradigms such as Petri net, object modeling, procedural programming, and so on [7–11]. Each approach provides different kinds of modeling strength with its own base formalism.

While previous research has mainly focused on developing new process modeling approaches, recent work aims at tailoring existing techniques to practical PSEEs. This current tendency reflects the research trend toward technology transfer and adoption. Considering this, it is necessary to look into PSEEs in terms of ordinary software systems.

A PSEE is a software system, and therefore, it has both functional and non-functional requirements. The common functional requirements are process modeling, analysis and enactment. In the past, the main issue in this field was functional capability. Therefore, various modeling formalisms were proposed. While the modeling formalisms themselves have important meaning and value, we cannot neglect the question of adoptability and usability in practice, which are usually characterized as non-functional requirements.
Recently, the usability of process modeling tools has become a new issue as pointed out in Ref. [12]. The formalities of the modeling techniques have inherent complexity in their representation schemes. For example, textual programming approaches require additional programming effort besides the actual development of target software systems. In the case of Petri-net-based approaches, the graphical representation might provide some easiness of modeling. However, Petri-net itself is still a formal expression. Therefore, it is not easily understood by ordinary users. Moreover, the net usually becomes extremely complex when detailed process activities are modeled.

Such problems result in communication difficulties among people in real situations, eventually causing difficulties in technology adoption. Moreover, communication difficulties result in many problems and delays during a software development. Thus, it would be much more helpful to have some means of representing processes in easier but unambiguous ways, while preserving the formality beneath.

In this paper, we introduce a software process design and management system, SoftPM (software process management system), which uses a high level Petri-net formalism for process modeling. To overcome the difficulty in using this formalism, SoftPM provides a multi-level modeling mechanism for the software process. It supports three different levels for process representation. They are cognitive, MAM-net and Pr/T-net levels. Each level has different roles in SoftPM: the cognitive level as the primary modeling layer, the MAM-net level as the core process representation formalism that we have proposed in Refs. [13,14] and the Pr/T-net level as the low-level base formalism. Our paper is organized as follows: Section 2 gives a brief assessment of existing approaches. In Section 3, we describe the overall mechanism of SoftPM in detail, as well as the basic rationale of our approach. In Section 4, we explain the transformation mechanisms for generating the corresponding MAM nets and eventually the Pr/T nets from a cognitive-level model. In Section 5, the analysis mechanism of SoftPM is explained. As an experimental study, we demonstrate the capability of SoftPM by simulating the ISPW-6 process example [15]. In Section 6, we describe the architecture of the SoftPM toolset and the process enactment mechanism. In Section 7, we describe the experience of applying SoftPM to a real world process. Finally, we conclude and discuss future work in Section 8.

2. Related works

Several techniques have been developed for process modeling in recent years. Since this area is very rapidly evolving, the underlying modeling paradigms are widely diverse. The most representative ones include the Petri-net-based paradigm [2,4], the rule-based [16], procedural programming [17], object-oriented [5,18,19] and system dynamics paradigms [1]. Each paradigm provides different advantages in representing software process. Among those paradigms, Petri-net-based modeling is noticeably a disciplined approach. With inherent modeling constructs (i.e. transitions, places, arcs and tokens), Petri net provides a powerful means of representing the static structure and the dynamic properties of software process. FUNSOFT nets [4,6] and SLANG [2,20,21] are the most representative approaches in the paradigm.

FUNSOFT net, as described in Refs. [4,6], is a high-level Petri net whose semantics are defined in terms of Pr/T Net [22]. After its initial prototype implementation called MELMAC, it has been reimplemented as a new product, and named LEU [12,23,24]. Unlike SoftPM, LEU requires users to design three separate models in order to represent process activity, artifacts and human resources. These three models are the activity model, the object model and the organizational model in the context of LEU. LEU uses FUNSOFT nets to describe activity models. The object model is a data model using an ER diagram-like format. The organization model is represented by hierarchical graphs representing divisions and personnel. While modeling each process element in different models has the advantage of providing a separate view for each element, it requires much more effort to model software process. Moreover, designing a concrete data (i.e. artifacts), model in the early process planning stage is a very difficult work, because concrete data models usually become available after the design phase of the usual software life cycle. Consequently, it becomes difficult to define accurately the relationship between the activity model and the data model. In terms of easiness in modeling, activity modeling still requires users to learn the Petri-net formalism. For process model analysis, this provides only an animated simulation and minimum manpower computation.

SLANG, which is used in a PSEE called SPADE, is a process modeling language based on a high-level timed Petri net called ER net [20,21,2,25]. For modeling software processes, SPADE provides several facilities including SLANG Editor, Object Editor, etc. as specified in Ref. [26]. SLANG is implemented on two different levels: Kernel SLANG and Full SLANG. Kernel SLANG is the lower level language which describes the basic Petri-net formalism, while Full SLANG describes process data and activities. In terms of the modeling mechanism, modeling process elements still remains as textual programming style. Looked at from the aspect of model analysis, no process analysis technique on the level of SLANG is explicitly proposed in Refs. [21,2,26]. However, several standard Petri-net analyses as well as time constraint analysis are supported at the level of underlying ER nets, but not in the semantics of a software process.

3. SoftPM approach

There are several characteristics of SoftPM that
distinguish it from the other approaches mentioned in the previous section. First, SoftPM relieves users of difficulties in understanding a complex modeling formalism (i.e. Petri net), while still preserving the strength of the formalism hidden underneath through the multi-level modeling approach. This is achieved through the multi-level modeling mechanism of SoftPM. Second, SoftPM supports the integrated modeling of process activities, artifacts and human resources in a single modeling formalism, MAM net. This integrated modeling approach not only removes the extra effort of modeling each element separately, but also supports the capability of managing the process through just a single model. Third, in order to aid managerial decision making, SoftPM offers a variety of managerial analysis techniques in addition to the conventional Petri-net analysis. Finally, the migration into heterogeneous platforms over the Internet can be easily achieved, since the SoftPM toolset is equipped with a Java-based thin client/server architecture.

Before explaining further about SoftPM, let us explore the underlying rationale of the multi-level modeling approach.

3.1. SoftPM rationale: multi-level modeling approach

SoftPM supports three different levels of process representation. They are cognitive, MAM-net, and Pr/T-net as shown in Fig. 1. Levels in this context mean different representation layers, not hierarchical abstractions. In SoftPM, a process designer initially models the target software process at the cognitive level using the supporting graphical editor. Then, the corresponding MAM-net model is automatically generated according to the transformation rules. Finally, the corresponding Pr/T net is also generated through the unfolding mechanism. There are several reasons for having these three different levels of representation.

First, it is too much to expect every process designer to be acquainted with the Petri-net formalism. In fact, the majority of process designers may not be familiar with the formal representation of Petri net. Such difficulty usually causes misunderstanding in the formal modeling constructs. To resolve this problem, SoftPM offers a cognitive-level modeling mechanism that has a style similar to those of ordinary data flow diagrams. By mainly interacting through the cognitive-level, process designers are freed from learning the complex Petri-net formalism for the process modeling.

Second, in order to take advantage of Petri-net formalism, we still need to preserve the Petri-net representation for the corresponding cognitive model. For this purpose, we use MAM net as the formal modeling mechanism in SoftPM. MAM net provides a powerful modeling mechanism for the software process domain. At the same time, MAM net offers better understandability than Pr/T nets through the support of high-level modeling constructs designed for the process elements. Although several research papers show some modeling examples using high level Petri nets, their basic modeling constructs are still limited to places, transitions and arcs. Real world process models usually become extremely complex forms when modeled with only such generic constructs. Consequently, the modeled process results in a spider-web look. Such graphical complexity decreases understandability, no matter how familiar users are with the formalism.

Third, to support the more detailed analysis of the process model, SoftPM uses Pr/T-net representation which is the base formalism of MAM nets. Any MAM net can be unfolded into an equivalent Pr/T net. In addition, this low-level representation also provides designers the chance to look at a process model with a very low-level formalism as low as other approaches.

Finally, the multi-level modeling approach yields a natural development process for the software process itself. Respecting the fundamental rationale of considering
software processes as software [27], we need to ask ourselves how many of us actually use formal methods in the initial modeling of a software system during the requirements analysis phase. When we develop a software system from scratch, we do not usually use formal methods during the initial modeling. Instead, cognitive methods such as the data flow diagram and object diagram are usually preferred. Formal methods such as Petri nets are commonly used for model verification and validation afterward. Indeed, the importance of process for modeling software process itself has been neglected. The most existing process modeling approaches have addressed only the functional effectiveness of their modeling techniques, not the process for software process modeling. The cognitive-level modeling in SoftPM not only offers a way of easy modeling, but also serves as a conceptual modeling tool during the early stages of software process development.

Soon after the three representations become ready, each representation is synchronized with the others for model modification, simulation and execution. Along with synchronization, SoftPM uses two different approaches to analysis. For structural analysis, conventional Petri-net analysis techniques such as deadlock and trap detections can be done at Pr/T-net level. For managerial analysis, SoftPM provides several useful analysis techniques at the MAM-net level. Before going into the detailed mechanisms of model transformation and analysis, we will explain the cognitive-level and the MAM-net level modeling mechanism in the following sections. We do not discuss the Pr/T-net level modeling mechanism in this paper, as Pr/T net formalism is a well-known Petri net class. Instead, we explain how a MAM net can be transformed into an equivalent Pr/T net in Section 4.

3.2. Cognitive level

The fundamental rationale for cognitive-level modeling is to specify all necessary information for the deterministic transformation into the corresponding MAM nets. In our approach, we use only two kinds of modeling constructs.
at the cognitive level to reduce user difficulty as much as possible. The two modeling constructs are process activity and artifact flow. As shown in Fig. 2, a process activity is denoted by a simple rectangle, and an artifact flow is denoted by an arc. Hierarchical activity abstraction is also supported.

With the graphical editor provided by SoftPM, users can easily design a target process model through simple drag-and-drop operations. Through the dialog-box-based transaction, as shown in Fig. 3, all necessary properties for each process activity can be defined and manipulated.

Using object-oriented paradigm for the implementation of SoftPM, each activity is modeled as an object. The following attributes are the properties of the process activity.

- **processName**: string
- **preRelation**: list of precedingActivity.output
- **postRelation**: list of succeedingActivity.input
- **expectedDuration**: int
- **actualDuration**: int
- **assignedAgents**: list
- **enactmentPolicy**: controlled/automatic
- **activityType**: atomic/abstracted
- **subActivityList**: null/list of subActivities
- **currentState**: notStarted/onExecution/partiallyFinished/Finished

A unique name for each process activity is represented by the attribute `processName`. The attributes `preRelation` and `postRelation` describe the relationships with other activities in terms of the execution order and the directions of the artifact transfers. The attribute `preRelation` holds the list of all preceding activities directly connected to the activity. Each entry of the list stores the name of a preceding activity with the output artifact needed from it. For the transfer of multiple artifacts from a single preceding activity, a separate entry is used for each distinct artifact. Similarly, `postRelation` holds the list of the succeeding activities and the artifacts to be transferred to them. The expected duration time of the activity is stored in `expectedDuration`. The expected duration of each process activity is prescriptively determinable at the project planning stage. However, it is usual to have some deviations during the actual process execution. At the completion of each activity, the actual duration is stored in `actualDuration`. The attribute `assignedAgent` holds the list of the human agents who are assigned to the activity. According to the enactment policy of each activity, its `enactmentPolicy` is set to either controlled or automatic. For the activity that needs permission from the manager for its enactment, its `enactmentPolicy` is set to controlled unless it is set to automatic. For the

![Fig. 4. ISPW-6 process example modeled on cognitive level.](image-url)
activity that does not have any sub-activities, its activityType is set to atomic, and subActivityList is set to null. If the activity is hierarchically abstracted, activityType is set to abstracted; subActivityList holds the list of its sub-activities. The attribute currentStatus indicates the execution status of the activity. Fig.4 shows the ISPW-6 software process example modeled on the cognitive level.

3.3. MAM-net level

MAM nets [13,14] the core modeling formalism of SoftPM, is a Petri-net-based process modeling formalism based on Pr/T net [22]. One of our concerns during the development of MAM nets is the integrated modeling of software process and organizational properties, specially the human agent. For efficient conducting of software development activities, the relationships between process model and organizational properties must be precisely analyzed [1,28]. Ordinarily, we have two independent representations for software process and organizational properties. The independent representations not only cause fundamental inefficiencies in analyzing the relational properties between the two models, but also may cause inconsistencies between them. Unfortunately, the existing Petri-net-based approaches do not support explicit modeling of both software process and organizational properties in a single representation. Moreover, even though the existing Petri-net-based approaches use high level Petri-net classes, the main modeling constructs are still restricted within the level of transition, places and arcs. This generic aspect of modeling constructs causes extreme complexity for modeling real world entities.

To resolve this problem, MAM nets provide modeling constructs for both the software process and human agent. MAM nets provide processStep for the straight mapping of a real process activity and AgentAllocationUnit (AAU) for modeling human resources.

3.3.1. Definition of MAM nets

A MAM net is defined as a tuple \( (PS, AAU, IP, OP, PI, AI, T, F, P; A, M_0) \), where \( PS = \{ps_1, ps_2, ..., ps_n\} \) is a finite set of processSteps: A processStep represents a process activity in software processes. Several processSteps can be hierarchically abstracted into one large processStep. The internal structure of the atomic (not abstracted) processStep is defined in terms of Pr/T nets as shown in Fig. 5(a). All the ports (i.e. the ellipses on the edges) are basically places. The transition on the left fires when the input artifacts arrive at the port \( ip \). This firing, results in placing an agent request in the port \( pq \) while transferring the input artifacts to the temporal place in the middle. When the requested process agents arrive at the port \( pc \) from the \( AAU \), the activity transition gets enabled and fires producing the output artifacts at the port \( op \), and releases the finished agents to the port \( pl \). As one can recognize, the transition on the right is the actual transition corresponding to the execution of the process activity. The arc expressions \( (i_a) \) and \( (o_a) \) imply the names of input and output artifacts, respectively, such as \( (source\_code) \) and \( (object\_code) \). The expression \( (a_c, pid) \) represents the code standing for the assigned process agents and the unique identifier for the processStep itself.

\( AAU = AgentAllocationUnit \): \( AAU \) represents a pool of the currently available human agents. The internal structure of the \( AAU \) is also depicted in Fig. 5(b). A processStep is supposed to send a request to the \( AAU \) in order to receive the assigned human agents for its execution. When an agent request arrives at the port \( aq \), the \( AAU \) first checks the request with the current available agents. If available, the requested agents are placed in the port \( as \), then the currently available agents are updated. If not available, the request is sent back to the port \( aq \), and waits until the requested agents become available. After the completion of the activity execution, the processStep restores the process agents back to the \( AAU \), so that they become available to other requests. When a processStep restores the finished agents,
the agents arrive at the port ar. Then, AAU updates the current available agents accordingly. 

The arcs in a MAM net are of four types classified by a connection role. For convenience, we call these four kinds of arcs, \( F_1, F_2, F_3 \) and \( F_4 \). \( F_1 \) is a set of arcs connecting the preceding processStep to the next one or a temporal place for a rework cycle. \( F_2 \) is involved in the processStep abstraction. In the processStep abstraction, the input artifacts of the abstract processStep must be distributed to its sub-processSteps. Also, the final output artifacts of the sub-processSteps must be merged to the output-artifact sending port of the abstract processStep. \( F_3 \) plays the role of distributing and merging these artifacts inside an abstract processStep. The arcs in \( F_3 \) are also involved in the processStep abstraction regarding the connections between the \( PI \) of the sub-processSteps and the \( PI \) of the abstracted processStep. The arcs in \( F_4 \) represent the connections between the AAU and processSteps regarding agent request, allocation and restoring. Thus, \( F \) is defined as follows:

\[
F \subseteq ( F_1 \cup F_2 \cup F_3 \cup F_4 )
\]

where

\[
F_1 \subseteq ( OP \times T ) \cup ( T \times IP ) \cup ( P \times T ) \cup ( TP )
\]

\[
F_2 \subseteq ( IP \times T ) \cup ( T \times P_{sub} ) \cup ( OP_{sub} \times T ) \cup ( T \times OP )
\]

\[
F_3 \subseteq ( P_{sub} \times T ) \cup ( T \times P ) \cup ( PC \times T ) \cup ( T \times P_{sub} ) \cup ( P_{sub} \times T ) \cup ( T \times PL )
\]

\[
F_4 \subseteq ( P \times T ) \cup ( T \times aq ) \cup ( as \times T ) \cup ( T \times PC ) \cup ( PL \times T ) \cup ( T \times ar )
\]

and \( F_1, F_2, F_3, \) and \( F_4 \) are exclusively disjoint to each other.

**3.3.2. Graphical notations and modeling mechanisms of MAM nets**

As shown in Fig. 6(a), a processStep is depicted as a rounded rectangle with the five ports as defined in the previous section. As shown in Fig. 6(b), the AAU is represented as a rounded rectangle with three communication ports on the top. The transitions, arcs and places of MAM nets use the same notations as ordinary Petri nets, as depicted in Fig. 6(c) and (d).
Fig. 7 presents examples of the basic process modeling mechanisms. In the case of two sequential process activities, the output-artifacts of preceding process activity are passed on to succeeding process activity as its input-artifacts. The two sequential processSteps can be modeled using two processSteps with arcs and a transition as shown in Fig. 7(a). Fig. 7(b) shows the case of a process activity that needs multiple input-artifacts from multiple preceding process activities. Fig. 7(c) shows the distribution of a single artifact to multiple succeeding processSteps. In order to provide a systematic abstraction mechanism of process activities, consistent connection mechanisms of the internal sub-processSteps are defined. As described in the definition of MAM nets, the internal connections of an abstract processStep can be modeled consistently regarding artifacts distribution, merging and agent allocation as shown in Fig. 7(d). One of the most important modeling mechanisms is the connection between the processSteps and AAU. This connection rule is defined statically. The modeling of the connection is trivial, as shown in Fig. 7(e). Therefore, the

![Diagram of process modeling mechanisms](image)

**Fig. 7.** Examples of basic modeling mechanism.

![Transformation rules: from cognitive model to MAM net](image)

**Fig. 8.** The transformation rules: from cognitive model to MAM net.
connection between the processSteps and AAU can be generated automatically without any human intervention, as in SoftPM.

4. Transformation mechanisms

In this section, we explain how the cognitive-level model is transformed into the corresponding MAM net, and eventually into the equivalent Pr/T net.

4.1. Transformation into MAM nets

Once a cognitive-level model has been designed by a user, SoftPM automatically generates the corresponding MAM net using the information given in the cognitive level. The generation of the corresponding MAM net is done using the transformation rules depicted in Fig. 8. The sequence of the generation basically follows the depth-first-search algorithm.

Fig. 9 shows the generated MAM net corresponding to the ISPW-6 software process example modeled at the cognitive level shown in Fig. 4. To distinguish graphically the connections between the processSteps and the AAU from those among the processSteps, we use dotted arcs and white transitions.

4.2. Transformation into Pr/T nets

The transformation of MAM nets into Pr/T nets is carried out by unfolding the net structures. Since the atomic processStep and AAU are defined in terms of Pr/T nets in their definitions, no unfolding is necessary for these atomic units. Hence, the actual transformation is the unfolding of the other units of MAM nets.

4.2.1. Unfolding overall net structures

Unfolding the overall net structures of a MAM net means the unfolding of the high level modeling constructs of the MAM net except for those atomic units described in the previous section. The high level modeling constructs are the ports (i.e. places) and the arcs. The places of a MAM net can hold more than one type of token. Also, the arcs of a MAM net can transfer more than one type of token. Based on the unfolding mechanism of high-level nets as described...
in Ref. [26], any modeled software process can be transformed into an equivalent Pr/T net without losing any properties. The net structures to be unfolded are of three types classified below:

- The ports $ip$ and $op$ of each non-atomic processStep
- The arcs involved in artifact transfers from and to non-atomic processSteps
- The port $pq$, $pc$, and $pl$ of each processStep, and the arcs and transitions between them and AAU

The ports $ip$ and $op$ of each non-atomic processStep: When a processStep is abstracted, it contains more than one sub-processStep. This implies that one or more types of token need to be placed in the ports $ip$ and $op$ of the abstracted processStep. Thus, the ports $ip$ and $op$ of the abstracted processSteps handle multiple tokens of different types. Based on the unfolding mechanism described in Ref. [26], the ports $ip$ and $op$ of the abstracted processStep can be unfolded by making individual places for each type of token.

The arcs involved artifact transfers from and to non-atomic processStep: These arcs can be classified into two kinds: those inside and those outside the abstracted processStep. The arcs inside the abstracted processSteps are the ones involved in the distribution of the input artifacts and merging of the output artifacts. The unfolding of these arcs is very straightforward. The transitions remain the same during unfolding. We just make separate arcs for each unfolded place of the ports $ip$ and $op$ of the abstract processStep. The arcs outside processSteps are the ones that make the connections between the port $op$ of a preceding processStep and the port $ip$ of a succeeding processStep. This unfolding procedure is also straightforward. Because the ports $ip$ and $op$ have been already unfolded, we only need to make separate arcs for each unfolded place.

The ports $pq$, $pc$, and $pl$ of each processStep and the arcs and transitions between them and AAU: Ports $pq$, $pc$ and $pl$ of each processStep hold only one type of token for each of them. So also the ports $aq$, $as$ and $ar$ of AAU. Therefore, we do not have to unfold the ports. And, each transition has exactly one input and one output place. Thus, each of the connections can be considered as a state machine that is an ordinary Petri net such that each transition has exactly one input and one output place. Using this property, we can unfold those parts by directly connecting each atomic processStep to the AAU using the reduction mechanism described in Refs. [29,30]. This reduction mechanism preserves liveness, safeness and boundedness.

After unfolding a MAM net into the equivalent Pr/T net, the conventional Petri-net-based structural analysis can be applied. The conventional analysis techniques are explained much in literature, so they are not discussed in this paper. Examples of the unfolding are shown in Fig. 10.

5. Managerial analysis

Besides the structural property analysis techniques of Petri net, SoftPM supports various useful analysis on the
dynamic properties of software process to aid managerial decision making during the process execution.

Most Petri-net-based approaches support structural property analysis techniques to detect the structural defects such as deadlocks, traps and liveness. In terms of the formalism of analysis, these conventional Petri-net-based analyses are quite powerful mechanisms. However, they have some shortcomings in terms of practicality. There exists a fundamental discrepancy between the formal analysis and the nature of the software process. While detecting errors through the formal analysis on a general software system can prevent potential runtime errors; this is not necessarily true for the software process domain. For instance, a deadlock detected by the Petri-net-based analysis does not absolutely imply a deadlock which will occur in actual process execution. Such discordance is due to the human-oriented nature of the software process. Therefore, the conventional Petri-net-based analysis is less meaningful in analyzing the dynamic properties of the process execution.

In practice, the actual execution of process activities can behave differently depending on the managerial strategy, even under the same process model. In the real world, the development environment is not guaranteed to be the same as initially expected. There can be some resource conflicts. Some activities can take much longer than expected, and so on. Considering these dynamic characteristics, it would be very helpful if we could have some mechanisms to handle these problems. Unfortunately, the existing Petri-net-based approaches do not provide explicit techniques for handling such dynamic characteristics of software process. For this purpose, SoftPM provides various managerial analysis techniques to extract useful information for efficiently conducting process activities.

5.1. Dynamic properties to analyze

There are many managerial elements related to the efficiency of conducting activities. In our approach, we focus on those elements related to activity duration and human resource utilization. SoftPM offers effective facilities for examining the following properties frequently considered by project managers in managerial decision making.

- Cumulative time consumption of each process activity
- Spots of agent conflict under a given schedule
- Influence of agent conflict
- Agent utilization
- Artifact idle time
- Concurrency of process activities
- Minimum manpower for project completion

5.2. Managerial analysis mechanisms

In the analysis of these properties, we assume that the expected duration of each process activity is determined during the project planning.

For our experimental study, we use the ISPW-6 software process example that we have used so far. The experimental environment setup is shown in Table 1. This example process consists of seven process activities since each of the original modify design and modify code activities is refined into its two sub-activities (i.e. modify design, into redesign and review design, modify code into edit code and compile and error check).

The managerial analysis mechanism is derived from timed Petri-net formalism. Based on the firing behavior of timed Petri net [31,29,32], we analyze the time-related behavior of MAM nets. In the MAM-net level process representation, the delayed transitions are the activity transitions of atomic processSteps. Thus, in MAM nets, the activity transition of each processStep has a positive delay time, and all the other transitions have zero delay time.

For our managerial analysis, we give some definitions, and derive equations for the time-related behavior of processSteps. The following derivations are done with the condition that all abstract processSteps are unfolded into atomic processSteps.

Let TC denote the time consumption of its argument. Let the activity transition of an atomic processStep $ps_i$ be denoted by $t_{ps_i}$. Each atomic processStep has exactly one single activity transition. Therefore, the time consumption of an atomic processStep is equal to the time consumption of its activity transition. Thus, the time consumption of each atomic processStep is defined as follows:

$$TC(ps_i) = TC(t_{ps_i})$$  \hspace{1cm} (1)

In the managerial analysis, we analyze the time-related behaviors of the process model in two different cases. In the first case, it is analyzed without considering the agent allocation (i.e. agent conflict). In the other case, we consider the agent allocation. We then compute the properties using the analysis results from the two cases. Before further derivation, we define the following attributes of the atomic processStep:

- $Input(ps_i)$: the set of input artifacts of processStep $ps_i$
- $Output(ps_i)$: the set of output artifacts of processStep $ps_i$
- $Prec(ps_i)$: the set of preceding processSteps of processStep $ps_i$
5.2.1. Without considering agent conflicts

The latest arrival time of input artifacts for processStep \( ps_i \) is the latest output producing time of its preceding processSteps. Thus, we define the latest arrival time of the input artifacts for processStep \( ps_i \), \( Input.arrival(ps_i) \), as follows:

\[
Input.arrival(ps_i) = \text{Max}\{\text{Finish}(ps_{p_c}) \mid (ps_{p_c}) \in \text{Prec}(ps_i)\}
\] (2)

If agent conflicts are not considered, the execution of a processStep is constrained only by the arrival status of its input artifacts. Therefore, its execution start time is equal to the latest arrival time of its input artifacts. Thus, the following equation holds.

\[
\text{Start}(ps_i) = \text{Input.arrival}(ps_i)
\] (3)

Using Eqs. (2) and (3), we derive the following Eq. (4):

\[
\text{Start}(ps_i) = \text{Max}\{\text{Finish}(ps_{p_c}) \mid (ps_{p_c}) \in \text{Prec}(ps_i)\}
\] (4)

The processStep finishes its execution when its output artifact is produced. Therefore, using Eqs. (1) and (3), we derive the following equation:

\[
\text{Finish}(ps_i) = \text{Start}(ps_i) + TC(ps_i)
\] (5)

Table 2 shows the ideal start and finish times of each processStep of the ISPW-6 example, obtained from Eqs. (4) and (5).

<table>
<thead>
<tr>
<th>Time</th>
<th>( ps_{sc} )</th>
<th>( ps_{sd} )</th>
<th>( ps_{se} )</th>
<th>( ps_{sc} )</th>
<th>( ps_{se} )</th>
<th>( ps_{sd} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>1</td>
<td>11</td>
<td>15</td>
<td>25</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>Finish</td>
<td>10</td>
<td>14</td>
<td>24</td>
<td>30</td>
<td>38</td>
<td>14</td>
</tr>
</tbody>
</table>

5.2.2. With considering agent conflicts

There can be two types of agent conflicts. One is where the requested agents were already allocated to other processSteps. The other is where multiple processSteps are requesting the same agent simultaneously. In the second case, a managerial scheduling policy is required. We use the shortest-job-first in our example. In Eq. (3), the start time of the individual processStep is determined only by the status of an input-artifact arrival. This is an optimistic expectation of a conflict-free agent allocation. Possible conflicts can be detected using Eq. (3) and \( Conflict.candidate(ps_i) \). We define the set of conflict processSteps with a processStep \( ps_i \) as follows:

\[
Conflict.set(ps_i) = \{ps_k \in Conflict.candidate(ps_i) \mid \text{Start}(ps_k) \leq \text{Start}(ps_i)\}
\]

In our example, we have the following conflicts:

\[
Conflict.set(ps_{rd}) = \{ps_{mt}\} \quad \text{and} \quad Conflict.set(ps_{mt}) = \{ps_{rd}\}
\]

Furthermore, the attributes \( Start \) and \( Agent \) of each processStep in \( Conflict.set(ps_i) \) describe when and to whom the conflict occurs. \( Input.arrival(ps_i) \) has the same properties as described in Eq. (2) because there are no additional constraints in the artifact transfer itself between any two processSteps according to the definitions of MAM nets. Thus, Eq. (2) is still valid in this case. However, the execution start time has different properties from the previous case. The activity transition in each atomic processStep is constrained by not only its input artifacts, but also by the arrival of the requested agents.

Since each processStep \( ps_i \) sends an agent-request token as soon as all necessary input artifacts arrive, \( Agent.request(ps_i) \) is defined as follows:

\[
Agent.request(ps_i) = Input.arrival(ps_i)
\] (6)

The execution of processStep \( ps_i \) starts only after both the necessary input artifacts and requested agent arrive. Thus, the start time of the actual execution of \( ps_i \) is defined as follows:

\[
\text{Start}(ps_i) = \text{Max}\{\text{Input.arrival}(ps_i), \text{Agent.arrival}(ps_i)\}
\] (7)

Therefore, by substituting \( Start(ps_i) \) of Eq. (5) with Eq. (7), the execution finish time of processStep \( ps_i \) is redefined as Eq. (8):

\[
\text{Finish}(ps_i) = \text{Max}\{\text{Input.arrival}(ps_i), \text{Agent.arrival}(ps_i)\} + TC(ps_i)
\] (8)

Considering agent conflicts, we can define the
Thus, the concurrency at time $t$ can be defined as Eq. (14). In the case of a conflict-free schedule, the ISPW-6 example has two processSteps running concurrently from the first day to the tenth. And, a single processStep runs for the rest of the period.

$$Concurrency(t) = \#\{ps_i \in PS \mid (Start(psi) \leq t) \land (Finish(psi) \geq t)\}$$

Finally, Eq. (15) represents the minimum manpower needed to carry out all process activities of the entire project by computing all manpower allocated to the project. In our example, we have 3.2 man-month.

$$Total\ Manpower = \sum_{all\ ps_i \in PS} TC(psi) \times \#Agent(psi)$$

6. Toolset architecture and process enactment

As depicted in Fig. 11, the SoftPM toolset consists of three subsystems. The SoftPM main system mainly supports the process modeling activity. The SoftPM enactment system carries out the enactment and monitoring of activities by interacting between the main system and the client system. The client system provides the functionalities of the artifact exchange and the actual activity execution through the Java-enabled web interface.

Process enactment in SoftPM is mainly driven in the unit of the processStep at the MAM-net level. Once the target process model has been set up, the actual process enactment takes place. Since SoftPM is designed and implemented using the object-oriented approach, most of the entities are implemented as objects including the process model, process activity, processStep, etc. The overall enactment flow is depicted in Fig. 11. The following procedure describes the enactment mechanism.

- A processStep is enacted. The enactment of a processStep can be initiated by either a project manager or an automatic operation depending on whether it needs the manager’s authorization or not.
- An active copy of the corresponding process activity is created in the form of a Java applet.
- The process activity is enacted by the processStep in the main system.
- The main system signals the corresponding agent through the enactment system.
- The agent accesses the activity applet using a web browser.
- The agent downloads the input artifacts from the centralized database through the browser. The corresponding tool is invoked either by plug-in or manually.
The agent completes the activity, accesses the activity applet, and updates the current status.

- The agent uploads the output artifacts.
- The activity applet updates the overall processes at the SoftPM main system via the enactment system.

7. Experiences summary

Developing a new technology always presents many lessons to be learned. In a pilot project to apply SofPM to a real world process, we applied SoftPM to a software development team at Korea Institute of Machinery and Materials. They develop embedded control systems for large-scale vehicles such as cargo ships and high-speed trains. During and after applying SoftPM, we made some preliminary conclusions.

In the matter of usability, our experience was very satisfactory. Since most project managers are familiar with the PERT-chart style of process modeling, the cognitive-level modeling of SoftPM turned out to be an effective approach.

Adopting process-driven development for a medium or small size organization is a burdensome task. Particularly, for those of low maturity, the heavy enforcement of process-driven activity often results in the early abandonment of the enforcement plan. Since the process execution mechanism of SoftPM did not require heavy work for the development members, the SoftPM environment was easily accepted by the members. What they actually needed was only to do their assigned jobs as notified by SoftPM, downloading and exchanging the related artifacts through SoftPM.

Moreover, the Java and Internet-based thin client/server architecture of SoftPM revealed another use. Unlike the groupware-based toolsets, it was easy to set up the PSEE with the flexibility of SoftPM. The Internet-based enactment mechanism of SoftPM was set up with the existing network environment. The participating members became quickly accustomed to the web-browser-based client interface.

Despite this encouraging experience, we found some other issues in PSEE. The most frequently asked question was how to determine the right granularity of process modeling. While process modeling in fine granularity yields more detailed activity control, it loses the ability to tolerate neglectable process deviations during the activity execution. The process deviation is an unavoidable fact during the process execution. Unfortunately, there are no concrete

![Fig. 11. The Toolset architecture and activity enactment flows.](image-url)
guidelines as yet proposed for modeling granularity in the research community.

Another frequently discussed issue was integration with the existing CASE tools. Our intended approach was to develop plug-ins for client web browsers to integrate SoftPM with existing tools. However, this approach turned out to be just an automatic invocation of CASE tools during the activity enactment. What the industry eventually needs is not only the activity integration, but also full integration of their tools through SoftPM. In the current state, we consider CORBA as a preliminary candidate solution for the integration platform.

8. Conclusion and future work

In this paper, we introduce SoftPM that we developed as a PSEE supporting tool. SoftPM is built on the top of MAM net which is a Petri-net-based formalism for modeling, analyzing and enacting processes.

Distinguished from other approaches, SoftPM smoothly complements the difficulties in using formal representation while preserving the power of the formalism beneath.

Besides, the underlying MAM net formalism supports the integrated modeling of software process and human resources. In addition, SoftPM provides various managerial analysis techniques. Moreover, SoftPM can be naturally ported into heterogeneous distributed environments with the aid of Java and the web-based thin client/server architecture. A screen shot of the SoftPM toolset modeling the ISPW-6 example is shown in Fig. 12. As a government-supported educational-industrial project, SoftPM will become a commercial product in the near future.

As future work, we are considering the following activities for the extension and evolution of SoftPM.

- Developing techniques to extract multiple views of the software process. These multiple views include the activity-oriented view, artifact-oriented view and agent-oriented view.
- Developing additional analysis techniques for each of the multiple views.
- Research on dynamic process evolution supporting process-change impact analysis.
- Extending MAM net properties to represent the impact of communication among agents.
• Developing an integration mechanism with various CASE tools.

References


