整合統一語言與派翠網路來建構工作流程
An Integrated Approach for Workflow Process Modeling and Analysis Using UML and Petri Nets

楊鎮華[1] 陳群奇[**]
Stephen J.H. Yang Chyun-Chyi Chen

摘要

工作流程管理系統是用來定義、協同、管理和執行複雜商業活動的一個軟體系統。此外，工作流程管理系統支援巨大且異質性分散的執行環境。在所有工作流程技術中，程序定義是工作流程管理系統的核心部分。基於工作流程管理組織的標準，我們提出以 UML 為程序定義方法。我們的方法包括了以下四個階段：應用使用情況圖來建構程序的功能需求；應用分類圖來建構系統資訊結構；應用活動圖來表示程序的控制流；和將使用情況圖轉換為活動圖。在本文中，我們將分類圖和活動圖轉變為顏色派翠網路和古典派翠網路。此外、派翠網路提供了許多的性質分析技術可以用來分析工作流程的正確性，像有限性、存活性、公平性和可到達性等，以及時序的性質，像終於、從今以後、直到等。

關鍵詞：工作流程、派翠網路、統一語言、程序定義、程序分析。

Abstract

Workflow management system is a software system that defines, coordinates, manages, and executes complex business activities. Furthermore, workflow management system supports large and heterogeneous distributed execution environments. Of all the workflow techniques, process definition is one of the kernel parts. Based on the workflow management coalition (WfMC) standard, we propose an UML approach for process definition. Our approach consists of four phases: Applying use case diagram for process functional requirements modeling; Applying class diagram for constructing process information structure; Applying activity diagram for process control flow; and transformation of use case diagram to activity diagram. In this paper, we then transform class, and activity diagrams into Coloured Petri nets, and classical Petri nets, respectively. In addition, Petri nets provide a variety of property analysis techniques for analyzing the correctness of workflow process, such as boundedness, liveness, fairness, and reachability etc and temporal properties, such as eventually, henceforth, until etc.

Keywords: Workflow, Petri nets, UML, process definition, process analysis.

1國立高雄第一科技大學電腦與通訊系

Department of Computer and Communication Engineering, National Kaohsiung First University of Science and Technology

**國立中央大學資訊工程所

Institute of Computer Science and Information Engineering, National Central University

1. Introduction

Many enterprises are facing pressures from market competition, reduction of cost, and rapid development of new products and services, they need new techniques to reduce processing time, allocate resource efficiently, improve performance, and shorten product's time to market. Workflow management techniques give the answer. However, Workflow management system is a software system that defines, coordinates, manages, and executes complex business activities. Furthermore, workflow management system supports large and heterogeneous distributed execution environments where sets of interrelated tasks can be carried out in an efficient and closely supervised fashion. For these characteristics, workflow management techniques has been applied comprehensively in many areas, such as telecommunication, manufacturing, finance, virtual enterprise, global logistics, business process reengineering, electronic commerce and so on.

With the evolution of the computer technology, workflow has experienced numbers of shifts in changes. In the early years, the process specification of workflow was hardcoded into application programs in order to satisfy certain requirements of office procedures. Nowadays, thanks to the progress of communication, information and object-oriented technologies, workflow management system has been able to support decentralized organizational units through graphical interfaces and a workflow engine to manage distributed tasks and resources on different locations (Veijalanen et al. 1995). Workflow has played an important role that provides back-end services to response front-end requirements in the age of electronic commence. Therefore, more and more venders have invested in the development of workflow products. These includes ActionWorkflow System of Action Technologies; IBM's Flow Mark; Visual WorkFlow of FileNet; InConcert produced by Xsoft (a division of Xerox Corp); FormFlow of Delrina; Regatta of Fujitsu (currently incorporated into ICL's TeamWARE); SAP Business Workflow by SAP; HP's WorkManager; OPEN/workflow of WANG and so on. However, many products are incompatible and no standards to enable these workflow products to work together. In 1993, the Workflow Management Coalition (WFMC) was established to encourage the development of workflow. WFMC provides a common "Reference Model" of workflow management systems to identify workflow management system's characteristics, terminology and components, and also enables individual specifications to be developed within the context of an overall model for workflow systems (Workflow Management Coalition 1994). Thus, all workflow products can achieve a level of interoperability through the use of common standard for various functions.

Most related researches of workflow could be classified into process definition modeling and analysis, activity coordinating and scheduling, workflow system architecture and design, and development methodology. Of all the workflow techniques, process definition is one of the kernel parts. It defines necessary information related to business process, such as the information of starting and completing conditions, constituent tasks, rules for navigating between activities, user tasks to be undertaken, applications that may be invoked and relevant data that may need to be referenced, and the resulting process definition will be executed by the workflow management system. Thus, the integrity and

accuracy of process definition will affect the result of execution. We will address our UML approach for workflow process definition in the following.

The rest of this paper is organized as follows. We will express what UML and business process are respectively in section 2. Section 3 represents how to utilize our UML approach to model business processes. Section 4 represents how to transform UML activity diagram and class diagram to classical petri nets and Coloured Petri nets. Section 5 introduces system properties, analysis methods, and the analysis of workflow properties. Section 6 concludes this paper with our future research.

2. UML Modeling and Business Process

In software life cycle, analysis phase is a major period to determine whether the software is corresponded to requirement of users. If the gap between domain experts or users and system developers is very narrow, the software system can be implemented to conform to the requirement of users. However, it is difficult to achieve the goal in the past. Because when the system developers receives the domain experts' description, they have trouble to catch the meaning of the terminology used by domain experts. Then, the system developers use their own system specification, such as specific specification language or unfriendly graphic representation used another terminology form a technical perspective. Further, the discrepancy from analysis to implementation results in that software system becomes difficult to use finally.

2.1 UML Modeling

In order to eliminate the difference between the business description and the software specification, unearthing common language understood by users and developers is imperative. Each symbol and semantic within the language must be defined clearly and intuitive for users. UML (Unified Modeling Language) is a well-defined and standard modeling language. UML consists of use case, sequence, collaboration, class, object, state, activity, component, and deployment diagrams (Rational and UML partners 1997). A system could be modeled via these diagrams form various aspects, such as structural, behavior, implementation, and environment views. Developers can design and exchange meaningful models without losing any information, adopted by software industry. UML furnishes users with user-friendly visual notations that improve the communications between users and developers, and translate the requirements of users into software specifications more precisely. UML even provides a unified development framework from analysis phase to implementation phase with software specifications. UML exhibits rich and expressive notations and semantic for specifying, visualizing, constructing and documenting software systems, business modeling and other non-software systems. Within UML modeling elements, some extended and tailored notations are suitable to represent the process definition of workflow. In the following, we will illustrate how to make use of UML approach to specify process definition. Firstly, we adopt use case diagram to express the specification of business functionality, goals, responsibility and interactions. Secondly, we adopt class diagram to express the organization of information related to business process. Finally, we adopt activity diagram to model business logical steps and dynamic behavior derived from previous use case diagram.

Before starting any steps of modeling, knowing what business processes can be implemented though workflow management techniques is necessary. Next, we would like to discuss what kinds of business process are suitable for workflow.

2.2 Business Process

In workflow management systems, business process is a set of one or more procedure(s) or activity(s), which realize business objectives or policy goals such as an insurance claims process, an order process, or a loan process. Even though workflow management techniques are able to reduce manual efforts and to provide enterprises with automatic environments, but these techniques may not be suitable for all business processes. Since the concept of workflow is originally used in solving management problems of business processes, it is adapted for a business process, whose activities are allocated, scheduled, routed, managed, and executed automatically. Business processes suitable for workflow management are usually characterized with properties such as automation, monitoring, repeatability, predictability, integration, and so on. In contrast, workflow management mechanism will not be suitable for business process characterized with simple, rarely used, or needs many manual works.

Once we can identify business processes suitable for workflow management techniques, the next issue is to decide using which method and how to model these business processes. In next section, we will present our UML through an illustration of a loan process of a bank.

3. Defining Business Process Using UML

We used to transform business processes directly into logical steps, such as Petri Nets, event flow, state transition diagram, etc (Aalst 1996; Lei et al. 1997). However, once processes change, we don't understand how logical steps are derived from the specification of a business process. Because such approaches only represent logical steps and lack for antecedent documentation. Therefore, in order to solve the above problems, we adopt UML approach consisting of use case, class, and activity diagrams to model diverse perspectives of business processes.

In the following, we will illustrate our UML approach through a banking loan example. There are steps for processing a loan for a typical banking system. The process contains interview with customers, accepting applications, creditability checking and pledge checking, evaluation, loan granted, and loan transfer finally.

3.1 Use Case Diagram and Business Processes

In order to capture the context of a business process, use case diagram is useful to represent goals, responsibility, functionality, and boundary intuitively. Use case diagram also expresses static interactions between business processes and their external objects. When notations of use case diagram maps into workflow mechanism, use case notations stand for sub-processes of a business process, and actor notations stand for participants (Workflow Management Coalition 1996). Therefore, based on the internal functions of a business process, each use case notation describes a sub-process, which composes the whole business process. Each use case also can be further detailed in another use case diagram. An actor of use case diagram may be

a user, an invoked application, a database, or a legacy system. Besides drawing uses cases or actors, a short textual description also helps readers understand the meaning of each use case and actor. Figure 1 shows a use case diagram for the loan process. This diagram contains five use cases and seven actors. In order to improve understanding, some textual descriptions for the content of each use case and actor are need. For example, the "interview" use case refers to that a staff contacts with a customer and collects related information filled in forms. The "accounting system" actor refers to that the external accounting system must be updated once the customer has obtained the loan.

Use case diagram is a helpful technique to exchange information. On one hand, it provides developers and users or domain experts with a high comprehensibility via intuitive notations and descriptions. On the other hand, it also furnishes a well documentation form the version perspective. After modeling, business processes will be turned into software specifications more precisely.

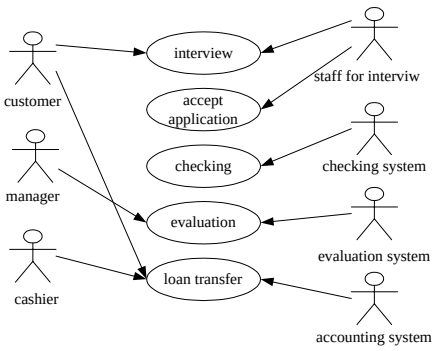

Figure 1. Use case diagram for the loan process

3.2 Class Diagram and Business Processes

From information aspect, class diagram of UML is useful to represent information of actors, roles, organizational units, and relevant data for business processes. These information objects can be seen as classes with relevant attributes in class diagram. In class diagram, a person may play one or more role(s) rendered by means of different classes according to his/her responsibilities. A class is an abstraction of description of a set of information objects from business processes. An attribute presents some properties within the information object and it usually displays enough information for the general readers to understand the meaning of the information object. If necessary, attributes of a class will be invoked as the process definition of the loan is executed and retrieves certain relevant data. For example, Figure 2 shows a part of information structure of the loan process including six classes: employee, worker, manager, application, pledge, and customer. For each class, it has its own attributes to express the intent of the class, such as the "employee" class contains id, name, and department attributes to exhibit the information of an employee. No class stands alone, each works in collaboration with others to describe relevant information about business processes. Hence, associations represent structural relationships between information objects. Association also represents both concepts of Aggregation and Generalization. These two special kinds of associations are beneficial in modeling information structure of business processes. Aggregation expresses a "whole/part" relationship, in which an information object of the whole has information objects of the part. For example, an application may hole one or more pledge(s) shown in Figure 2. Generalization expresses an "inheritance" relationship between a general information object and a more specific information object,

in which a specific information object can inherit properties of a general information object. For example, a worker or a manager can inherit all properties defined by the "employee" class shown in Figure 2. Furthermore, association also has a multiplicity value indicating how many instances of class may be linked to an instance of another class. For example, the "application" class has a one-to-many association to the "pledge" class referring to that at least one instance of the "pledge" class is owned by one instance of the "application" class shown in Figure 2.

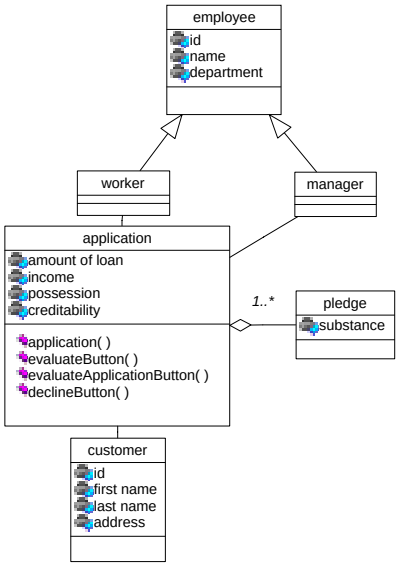

Figure 2. Class diagram for the loan process

3.3 Activity Diagram and Workflow Primitives

Even though use case diagram represents business processes, it cannot show the order of each use case instance and dynamic behavior. Within the UML model elements, both sequence diagram and activity diagram support to describe the dynamic behavior of use cases. Whereas sequence diagram emphasizes the flow of control from object to object, activity diagram emphasizes the flow of control form activity to activity (Booch et al. 1996). In contrast to sequence diagram, activity diagram is very useful in modeling the process definition of the workflow and in describing the behavior that contains a lot of parallel processing. Each activity can be followed by another activity. Unlike the flow chart, the activity diagram not only represents simply sequencing but also can direct parallel processing. This is essential for business processes. In order to improve efficiency, many tasks must be processed simultaneously within business processes.

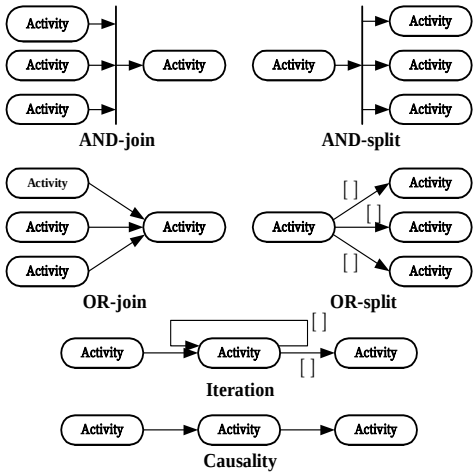

Figure 3. Workflow primitives specified by activity diagram

WFMC defined six primitives to model business logical steps (Workflow Management Coalition 1994). In this paper, we adopt activity diagram to specify these six primitives because activity diagram supports the modeling of workflow activity, transition, condition, synchronization, parallelism, iteration, etc. We specify workflow activity

by means of activity notation of activity diagram and workflow transition by means of transition notation with an arrow of activity diagram. Figure 3 shows how activity diagrams are corresponded to the six workflow primitives defined by WFMC. AND-join primitive expresses that two or more parallel threads meet into a single thread and the synchronization bar may only be crossed to next workflow activity when all input transitions on the bar have been triggered (Muller 1997). AND-split primitive expresses that a single thread split into two or more threads and the output transitions attached to the synchronization bar are triggered simultaneously. OR-join primitive expresses that when two or more alternative workflow branches re-converge into a single thread without any synchronization. OR-Split primitive expresses that when a single thread makes a decision upon which branch to take when encountered with multiple workflow branches. Branches between activities can be guarded by conditions. If guards validate, the transitions close to them are triggered to next workflow activities. Iteration primitive expresses that a workflow activity cycle involves the repetitive execution of workflow activity until a condition is met. Causality primitive expresses that two or more workflow activities are executed in a sequential form without any join or split.

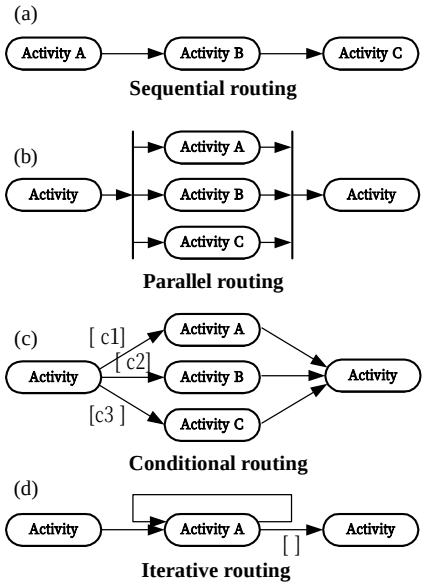

Figure 4. Process routing presented by workflow primitives

With the above six workflow primitives specified by activity diagram as shown in Figure 3, we can further to define four process routing, which are sequential, conditional, parallel, and iterative routing (Aalst 1996; Lawrence 1997). In workflow process, the four routing can be used to model any business process workflow and business process workflow can be used to model enterprise workflow. The results are show in Figure 4. Sequential routing is used to deal with causal relationships between activities. For example, three activities A, B, and C are executed sequentially. Figure 4.a shows how to use Causality workflow primitive to model sequential routing. Parallel routing is used when the ordering of activity execution is not of concern. For example, three activities A, B, and C are executed and the order of their execution is arbitrary. Figure 4.b shows how to use AND-split and AND-join workflow primitives to model parallel routing.

Conditional routing is used when instances need to be considered and those instances may depend on the workflow attributes. For example, in Figure 4.c one of three activities A, B, and C are executed and one of execution is depend on the workflow attributes whether satisfy condition c1, c2 and c3. Figure 4.c shows how to use OR-split and OR-join workflow primitives to model conditional routing. Iterative routing is used to deal with activity which need to execute one or more than one times. Figure 4.d shows how to use iteration workflow primitive to model iterative routing.

3.4 Transformation from Use Case to Activity Diagrams

Activity diagram allows the representation of logical steps of a business process for use case diagram. However, we have to know how to transform use case diagram into activity diagram. Before transformation, scenario is an advantageous mechanism to help developers understand procedures of a business process from starting to ending. Scenario is an instance of a use case that describes how use case is realized. It is a course of the flow of events for a use case, and contains preconditions, a primary scenario and one or more exceptional scenario(s). Developers can unearth objects from scenario through typical UML approach for OO modeling. In our approach, we adopt similar manners discussed previously, but we find activities applied in workflow process definition from scenarios and not objects. For example, the scenario of the “evaluation” use case shown in Figure 1 indicates a precondition describing an application have been received to start with evaluation, a primary scenario describing a evaluation system received a customer’s application and dispatched to junior officers or senior officers to review based on the amount of the application, and finally the application may be accepted or declined. In the following, we will discuss the transformation of use case diagram to activity diagram.

Firstly, it is necessary to identify the preconditions of initial state and the postconditions of final state, which betters to comprehend the border of the flow of control. Then, using scenarios to work through it can help identify activities of business processes and transitions from activity to activity. Before distinguishing the type of transitions, it is not hard to get sequential and branching transitions via scenarios in advance and then consider forking and joining transitions. If a transition meets a branching, guards should not overlap and must cover all possibilities. Similar to use case diagram, a complicated activity can detail further in another activity diagram. For example, Figure 5 shows the loan process specified by activity diagram. These activities are derived from the scenarios of use case diagram shown in Figure 1. The process starts as a customer applies a loan and ends as the loan of the application has been transferred or declined. It is difficult to specify forking and joining of parallel transitions at first time, such as the “creditability checking” activity and the “pledge checking” activity. Hence, we can consider the flow from the “creditability checking” activity to the “pledge checking” activity or from the “pledge checking” activity to the “pledge checking” activity in advance and then further specify the parallel processing.

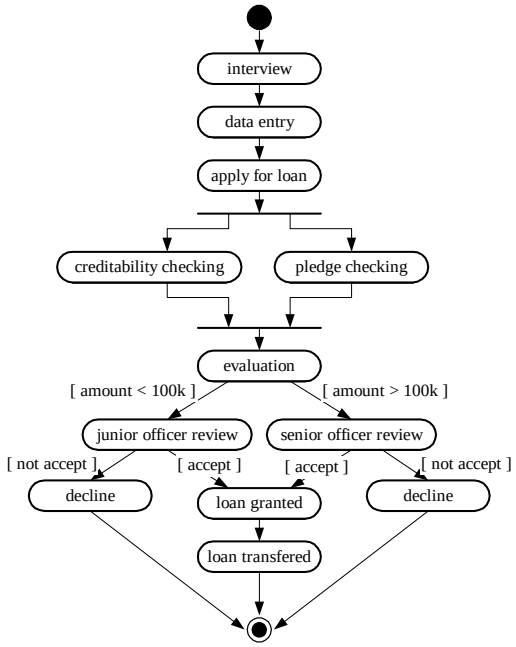

Figure 5. Activity diagram for the loan process

In order to follow the standard defined by WFMC, activity diagram must be decomposed to correspond with the six primitives of workflow. Based on previous discussion about activity diagram and workflow primitives, Figure 5 can be decomposed as shown in Figure 6. Compared with the six primitives as shown in Figure 3, we can see that Figure 6.a and Figure 6.b are Causality primitive. Figure 6.c is an AND-split primitive. Figure 6.d is an AND-join primitive. Figure 6.e, Figure 6.f, and Figure 6.g are OR-split primitive. Figure 6.h is an OR-join primitive.

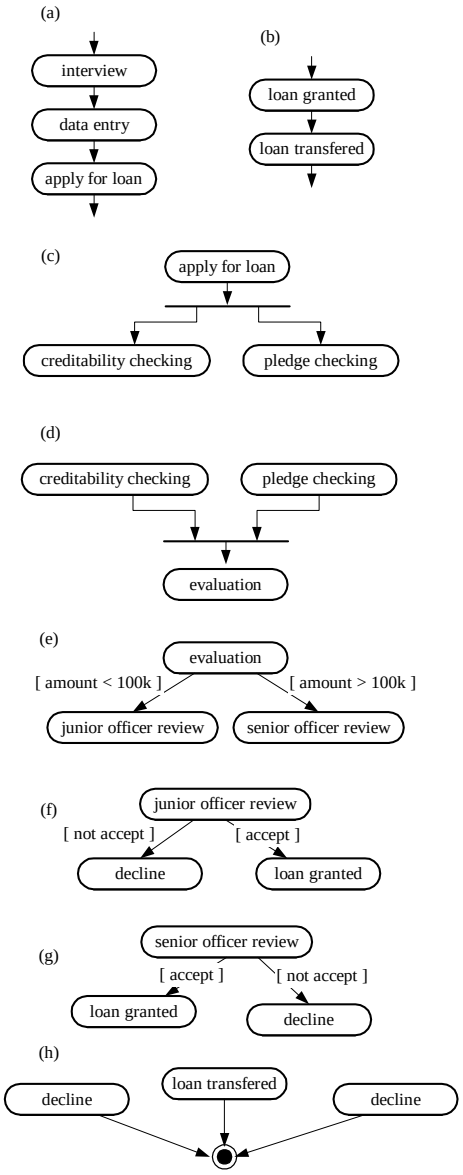

Figure 6. Decomposed activity diagram of Figure 5

4. Transformation from UML to Petri Nets

We use UML approach to model

business process that solve the following problem by logical steps modeling such as Petri Nets, event flow, state transition diagram, etc (Aalst 1996; Lei et al. 1997). Once processes change, we don't understand how logical steps are derived from the specification of a business process. Because logical steps approaches only represent logical steps and lack for antecedent documentation. But UML approach model lack business process analysis. In order to perform business process analysis, we transfer UML to Petri nets. This because Petri nets provide many mathematical formalism for properties analysis, which can be used to analyze the correctness of workflow process definition. In this section, we discuss how to transfer UML activity diagram and class diagram into classical Petri nets and Coloured Petri nets.

4.1 Transformation of UML Activity Diagram to Petri Nets

In this subsection, we address the rules that transform activity diagrams into classical Petri nets. Peterson (Peterson 1981) presents the transformation of flowcharts to classical Petri nets. The notion of flowcharts is similar to the one of activity diagrams (Boocks 1998). Thus, we adopt his idea from the flowcharts to classical Petri nets and refine them as follows. Figure 7 shows the corresponding Petri net transformed from the activity diagram in Figure 5. A state is called a source state that the state will change to other states if an event occurs. A state is called an activity state if we can further divide it into many states. A state is called an action state if we can't further divide it. Therefore, an action state is atomic and an activity state may be composed of one or more action states. In other words, an action state is a special case of an activity state. In addition, if we desire to understand the details of an activity state, we can zoom into the contents of the activity state via another activity diagram. An activity diagram would seem to be very analogous to a Petri net. Therefore, the suitable transformation from activity diagrams to Petri nets replaces the vertices of activity diagrams with transitions in Petri nets and the arcs of activity diagrams with places in Petri nets. The vertices of activity diagrams are represented in different ways, depending on the class of the vertices, that is, action state or branch. Figure 8 shows the two ways of transformation.

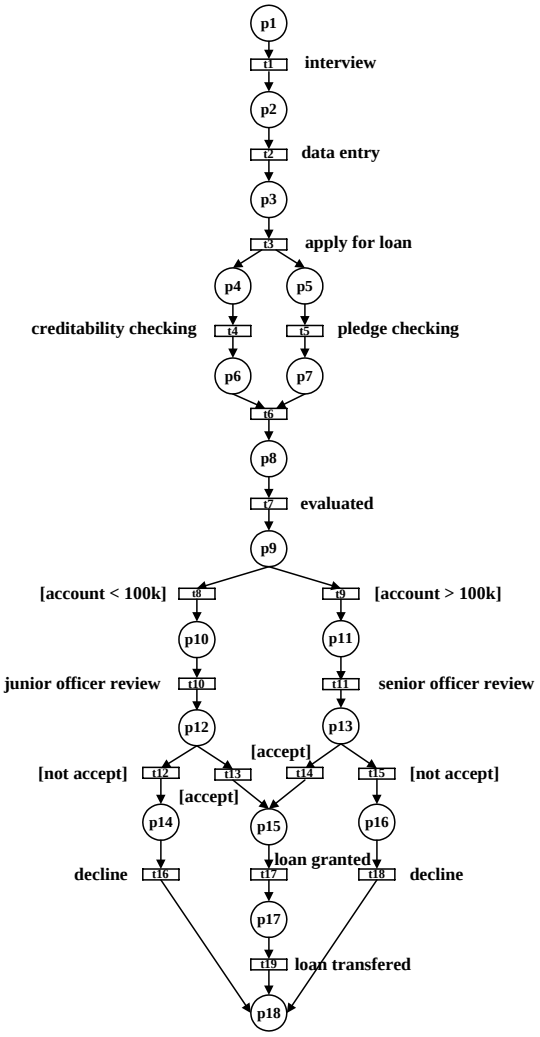

Figure 7. A Petri net model transformed from the activity diagram in Figure 5.

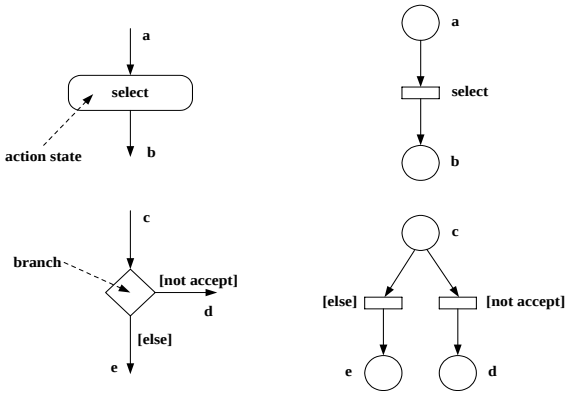

Figure 8. Transforming action state and branch vertices in activity diagrams to transitions in Peri nets

In UML, we usually use a synchronization bar to model the forking and joining of control. Similarly, we can specify these operations with transitions of Petri nets. Figure 9 illustrates fork and join operations of activity diagrams transformed to the transitions of Petri nets.

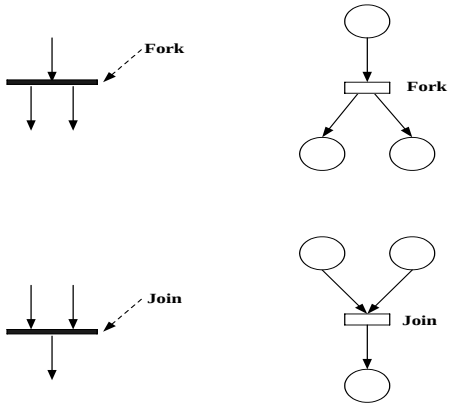

Figure 9. Transforming activity diagrams' fork and join operations to Petri nets' transitions

4.2 Transformation of UML Class Diagram to Petri Nets

This subsection deals with how class diagrams are transformed into Colorured Petri nets. Watanabe, et al. (Watanabe at el. 1998) integrate class diagrams and statechart diagrams in order to obtain Coloured Petri nets and then verify its correctness for specification before implementation in Java. Hence, we, in detail, address transformation in this subsection and add some stuff about analysis of Coloured Petri nets in the following section. Figure 11 shows a Colored Petri net transformed from the class diagram in Figure 2 and the statechart diagram in Figure 10. A class can be treated as a set of attribute and operations. The value of a token can stand for the value of an attribute of an object, so that we can stand for a class with a set of Coloured Petri nets where each Coloured Petri net represents its corresponding operation in the class. Therefore, we first transform the operation of a class into Coloured Petri nets and then integrate the Coloured Petri nets based on their relationships, such as method invocation (aggregation) and generalization between classes. Table 1 gives the itemize the transformation between Object-Oriented concepts and Coloured Petri nets notation.

Table 1. Transformation between OO and Coloured Petri nets

Object-Oriented concepts	**Coloured Petri Nets**
Class	A group of Coloured Petri nets
Operation	Coloured Petri nets
Object (instance)	Token associated with an object identifier (class name) and an identifier of a thread
Thread	The same as above
Attribute	Place
Value (inputs, outputs of operations, and attribute value.	Token associated with related color
Type of Values	Color
Action	Transition

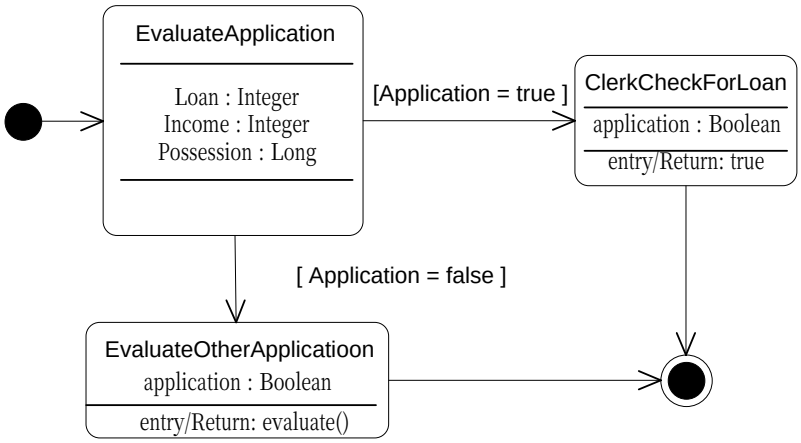

Figure 10. A statechart diagram for Application class

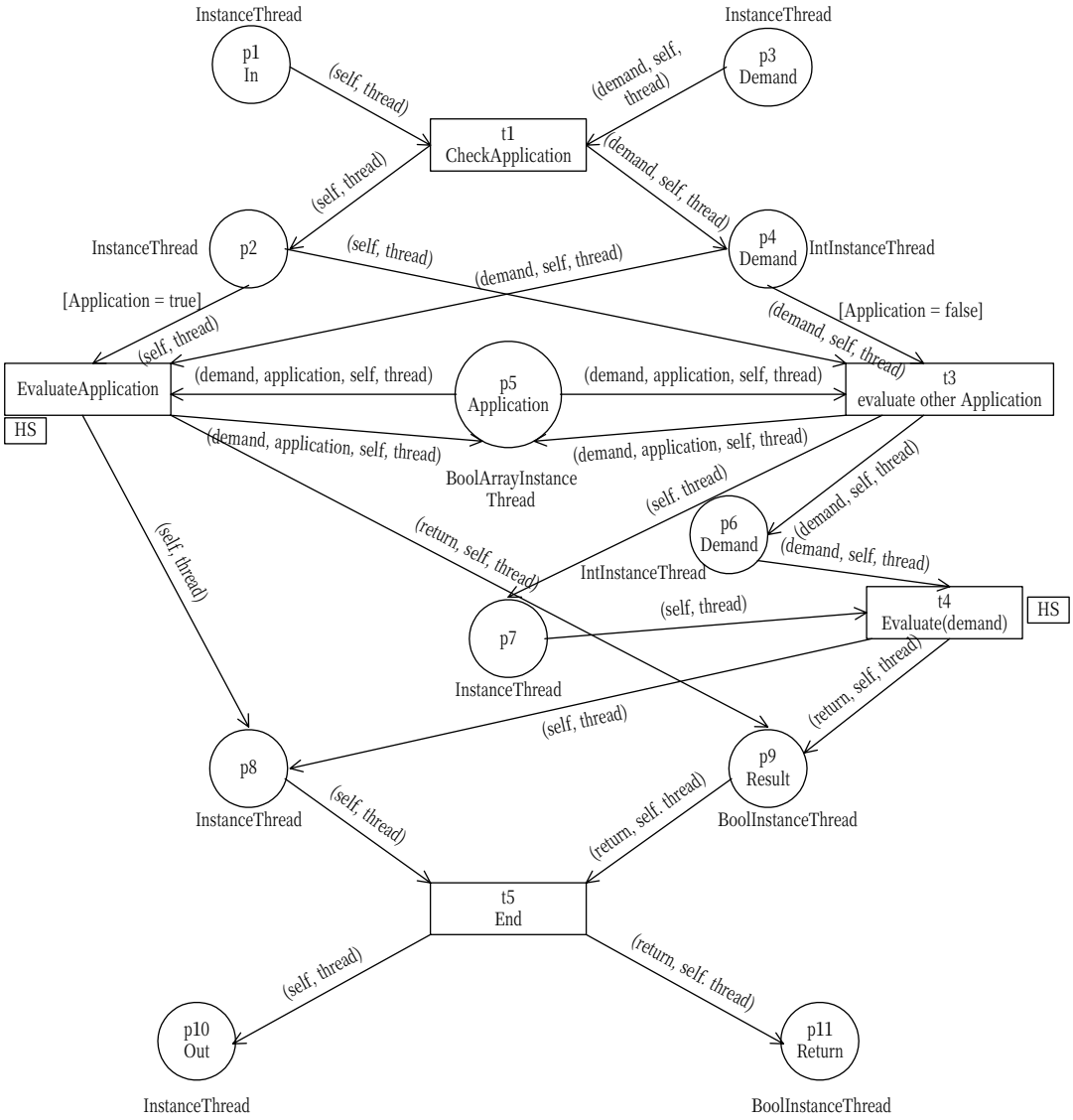

Figure 11. A Coloured Petri net transformed from Figures 6 and 8

4.2.1 Outline of Transformation

As Figure 12 shows, because of the sophisticated structure of class diagrams, for the transformation between Petri nets and class diagrams we briefly give a big picture and itemize the procedure of the transformation of class diagrams and statechart diagrams to a Coloured Petri net as follows.

1. Transforming attributes and operations,

which are specified in class diagrams and statechart diagrams, in Figure 2 between classes, into a Coloured Petri nets.

2. Based on the relationships generalization deriving coloured Petri nets (see the case Coloured Petri net/Relatoinship). Those relationships can be elicited from the class diagram.
3. According to the relationships aggregation, integrating the above two Coloured Petri nets into one coloured Petri nets.

We will apply the example bank loan process to illustrate the above procedure in the following subsections.

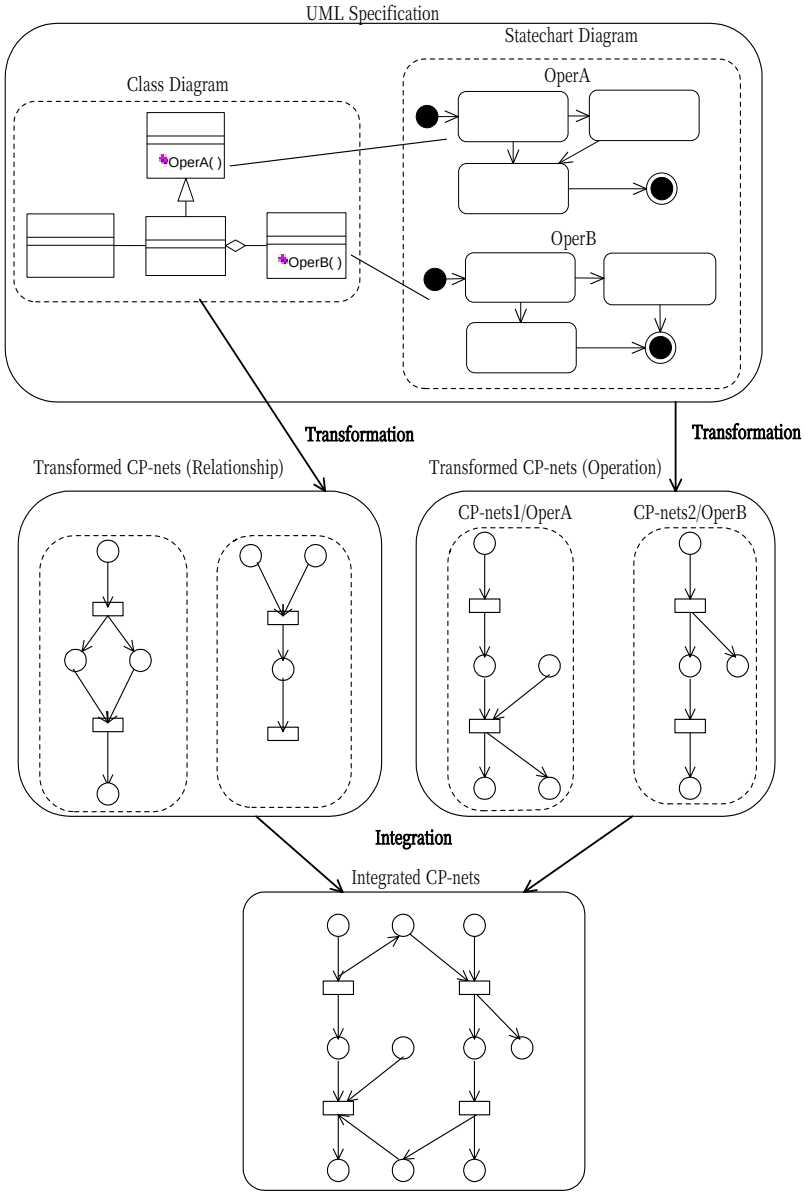

Figure 12. Transformation of class diagrams and statechart diagrams to a Coloured Petri net

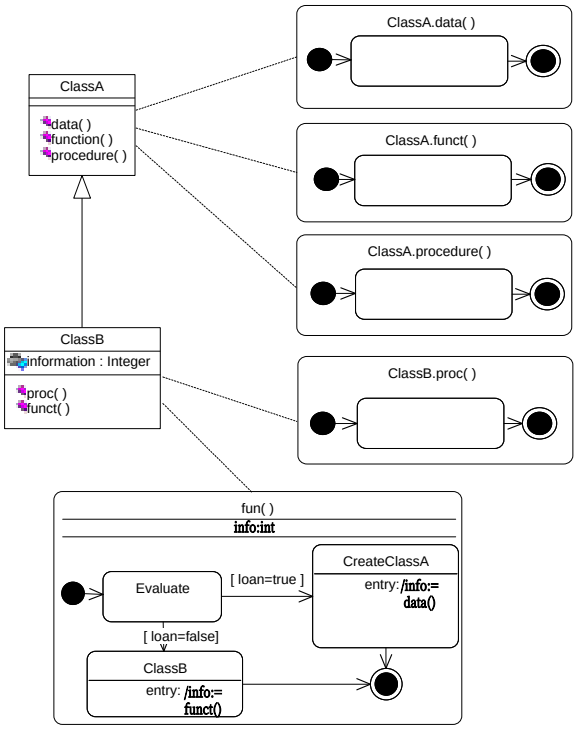

Figure 13. A class diagram with statechart diagrams indicating its properties and behavior

4.2.2 Transformation of Methods

Figure 14 shows the CP-nets transformed from the operation “data( )” of class “ClassA” in Figure 13. Since there is only a state within the statechart diagram connected with the class “data( )” and the operation “data( )” has no argument, the structure of the transformed CP-nets is very simple. In the Figure, there are two places “entry” and “exit” to be introduced as the starting point and ending point of the operation “data( )”, respectively. Once a token flows into the place “entry”, the transition “ClassA.data( )” is capable of firing and will initiate the process. If the transition “End” is fired, then produces a token in the place “exit”, which shows that the process ends. When “t1” in Figure 14 fires, the state of “data( )” in the statechart diagram of Figure 10 change imediately. Each token before going through an arc is associated with a pair of a object identifier and a thread identifier denoted as “objID” and “threadID”, respectively. It shows that when a token is flowing on an arc, the “objID” and “threadID” will be replaced with a object identifier and a thread identifier of the token, and those two identifiers can’t be changed during flowing. That is, a token which stands for an object of a class will not change its identifier from the starting point to the ending point of an arc, and object identifier and thread identifier represent where the token is produced and used. Transforming the operation with arguments into CP-nets is more complex than the above. Figure 25 shows the CP-nets which is a template for

representing the operation transformed with arguments. In the figure, token, which differs from other token in Figure 14, has many attributes that represent the type of different information such as object identifier, thread identifier, arguments, or return values. In addition, a transition “End” is inserted into the end, in order to ensure that the object that invokes the operation and assigns the parameters is identical, and then decide if pass the return values to it.

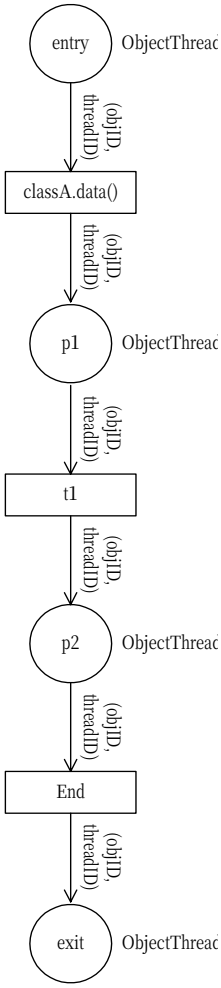

Figure 14. Transformation of Operation

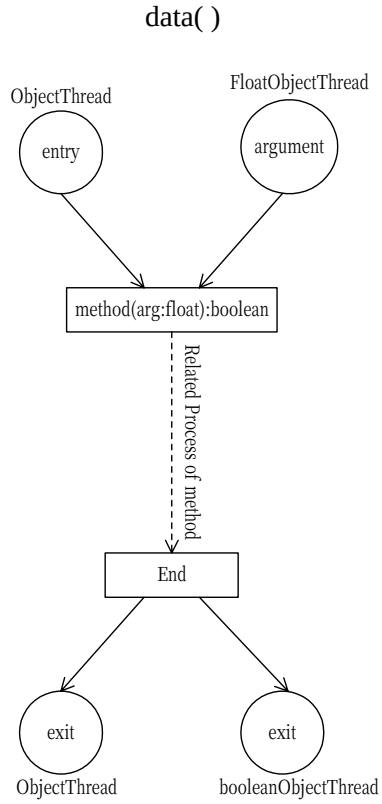

Figure 15. A method invocation

4.2.3 Transformation of Object Creation

The transformation of the constructor operation is analogous to that of the ordinary operation. As shown in Figure 16, the only difference is that a module “identifier creator” is introduced into the entry of the process as a device, in order to create an object identifier (a pair of class name and an integer). That is, the transition asks the CP-nets with the places ”ClassName” and “ID” as its input places in the box “InstanceCreator” for producing a token with a pair of ClassName and integer (ID) in the place “exit”. The token with object identifier is the output of the CP-net “InstanceCreator”, and the ID number will be +1 to be the ID number of next object

identifier to be created.

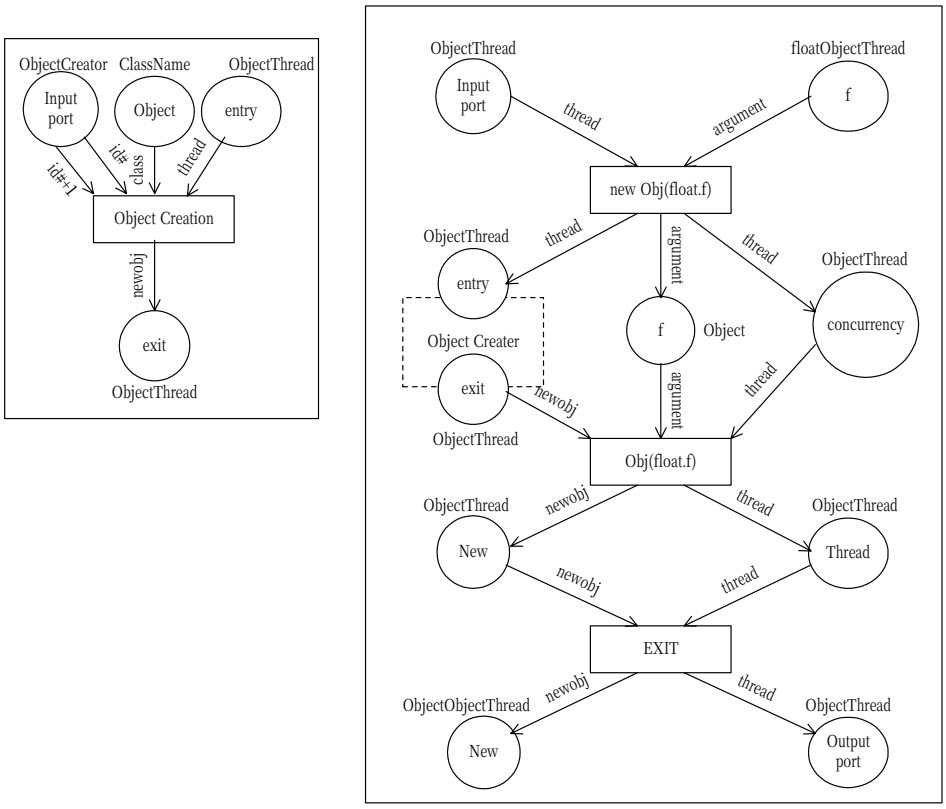

Figure 16. A Coloured Petri net for Creating Objects

4.2.4 Inheritance and Polymorphism

The way of transformation of inheritance and polymorphism mechanism is to find which operation is invoked (i.e., polymorphism) and which class the invoked operation belongs to (i.e., inheritance). Figure 17 and 18 illustrate the transformation of how to find the operation and the class (as mentions above) according to the two mechanisms.

Also, a transition whose name is identical to the operation in class diagrams is inserted into the starting point of CP-nets as the beginning of the invocation. For example, the transition “I1.inheritance( )” denotes the beginning of “I1.inhteritance( )” method invocation. If there are two or more operations associated with the transition, the expression should be replaced with guard expressions. For example, if a guard expression is “[A.method(), B.method( )], it stands for that the operations “A.method( )” and “B.method( )” must be invoked simultaneously. When transforming the operation by means of inheritance mechanism, the distance between the class that the object belongs to and the class that the operation belongs to is derived in the hierarchy of inheritance (generalization/specification)

(Booch et al. 1998), and then based on the distance decide an arc linked with the nearest superclass and attach the guard expression to the transition. For example, in Figure 17, if a token whose class name is either "I1" or "I2", and then transition "I1.inheritance( )" that the method is transformed into will be fired (i.e., I1.inheritance( ) will be invoked) through the inheritance hierarchy in the left of Figure 17. Likewise, if a token with either class name "I3" or "I4" is in the place "Input port", and then transition "I3.inheritance( )" that the method of class I3 is transformed into will be fired (i.e., I3.inheritance( ) will be invoked). The colors of the place "entry" include "threadID" and "objectID" which are class name and ID, respectively. In addition, the concept of polymorphism is analogous to that of the above. The arc expression is used to check if the token equals the class that object belongs to. For example, If the class name of the token of the place "p" is "Poly1", then the system will fire the transition "Poly1.polymorp( )" or else fire "Poly2.polymorp( )" if "ClassName = Poly2".

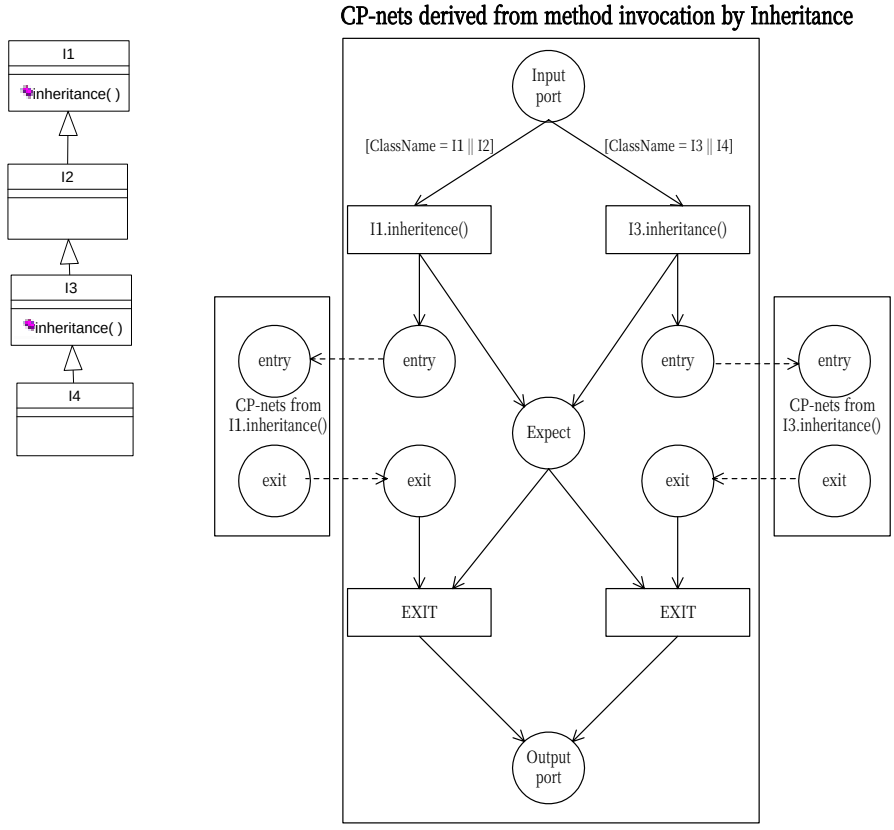

Figure 17. A Coloured Petri net based on Inheritance mechanism

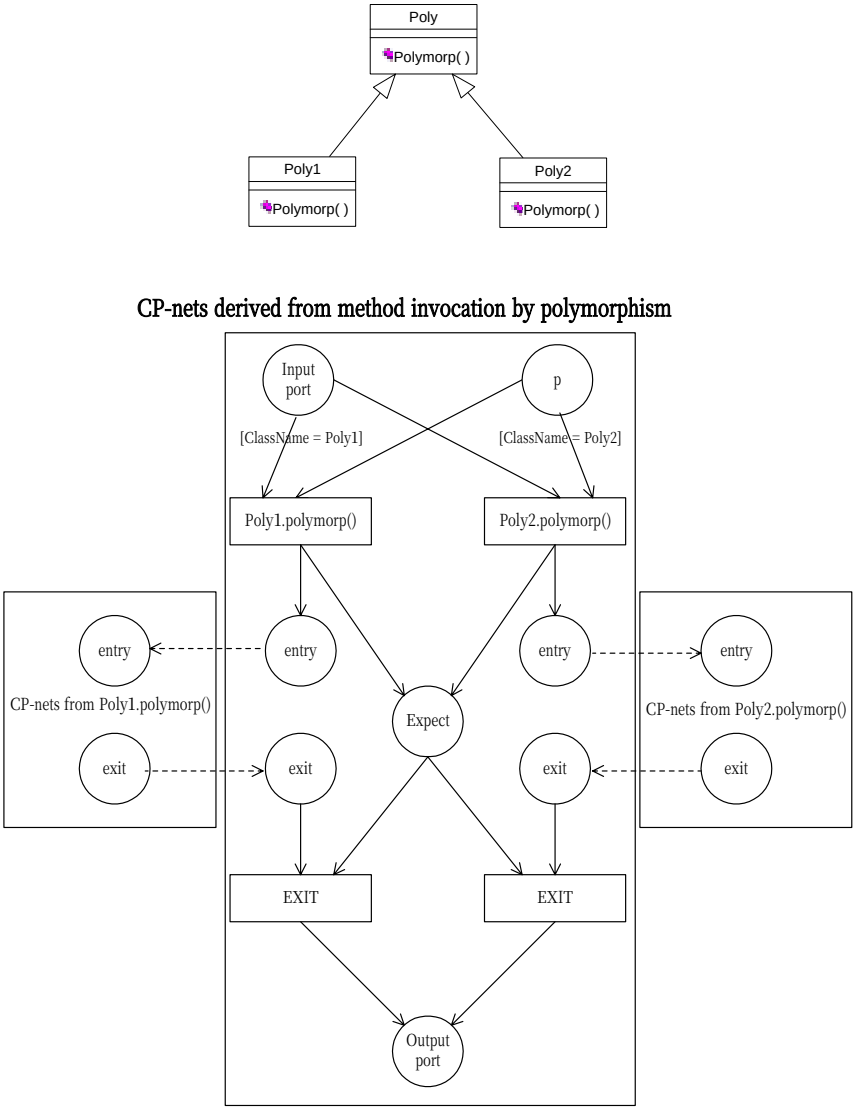

Figure 18. A Coloured Petri net in terms of polymorphism mechanism

5. Workflow Process Analysis

In this section, we describe how to analyze the properties of transformed Petri nets. Software consists of data, function, and behavior (Brooks 1986), and requirements can be classified as functional requirements. Thus once a system had been constructed, we shall require the system meet the functional requirements (i.e. data, function, and behavior). Therefore, we induce the relationships between our Petri net models and software. That is, data and functions correspond to Coloured Petri nets, and behaviors correspond to classical Petri nets. The correspondences are summarized below

in Table 2.

Table 2. The relationships among Petri nets, UML and requirements

Requirements	Functional		
	Data	Functions	Behavior
Petri nets	Coloured Petri nets	Coloured Petri nets	Classical Petri nets
UML	Class diagrams	Statechart diagrams	Activity diagrams

From the above inductions, we determine to focus our analysis on three views, i.e., behavior (classical Petri nets), data (Coloured Petri nets) and function (Colured Petri nets) aspects. In case we found properties are not satisified, we make modification directly on the corresponding Petri nets. The modification continues until the modified Petri nets can satisfy the specified properties. In the above subsection, we first introduce system properties and analysis methods of Petri nets, then we begin our Petri nets analysis from classical Petri nets to analyze behavioral properties.

5.1 System Properties

In this subsection, we address behavioral properties, and temporal behaviors from the viewpoints of Petri nets and temporal logic. We first classify behavioral properties into reachablility, conservation, boundedness, safe, liveness, reversibility, home state, persistence, synchronic distance, and fairness. Secondly, we address the reasoning of temporal behaviors such as next, eventuality, always, and until.

5.1.1 Behavioral Properties

Peterson (Peterson 1981) originally addressed the following behavioral properties. A rachability problem is a problem of finding *M* if $M \in R(M_0)$ for a given marking M in a net (N, M_0), where R(M_0) means the set of all possible markings reachable from M_0 in the net (N, M_0). A Petri net (N, M_0) is said to be strictly conservative if, for all reachable marking from M_0, the total number of tokens of each reachable marking is equal to that of tokens of M_0. A Petri net is said to be bounded if and only if for each place p of any marking reachable from M_0 such that the number of tokens is finite and less than k, where k is a finite number. A Petri net is said to be safe if and only if it is 1-bounded, i.e. for each place of the token number is less than 1. We will define five different levels of liveness as follows. For a set of all possible firing sequences from M_0, a transition t is said to be L0-Live (dead) if t can never fired in any firing sequence. t is said to be L1-Live (potentially live) if t can fire at least once in some firing sequences. t is said to be L2-Live if t can fire at least k times in some firing sequences and k is given any positive integer. t is said to be L3-Live if t can fire at infinitely often in some firing sequences. t is said to be L4-Live (Live) if t is L1-Live for every markings. A Petri net is said to be reversible, if it can always get back to the M_0. A marking M is said to be a home state, if for any marking M' in R(M_0), such that M is reachable from M'. A Petri net is said to be persistent, if for any two enabled transitions such that the firing of one transition will not

disable the other. Synchronic distance is the maximum difference of firing counts of two transitions t1 and t2, i.e., d12 = max |F(t1) - F(t2)|, where F is the number of times that transition ti, i = 1,2 fires in F. Two transitions t1 and t2 are said to be in a bounded-fair (or B-fair) relation if the maximum number of times that either one can fire while the other is not firing is bounded.

5.1.2 Temporal Behavio**r**

Even thought the capability of representing timing constraints, TPN is lack of the expressive power of certain temporal behaviors, such as "t1 will fire next", "Eventually t1 will fire", and "t2 cannot fire unless t1 fire". As a result, temporal PN is proposed to solve the above problems (Suzuki and Lu 1989). Let σ is an infinite firing sequence starting from a current marking Mc, proposition pj is defined as a place pj having at least one token; proposition tj is defined as a transition tj is fireable. The temporal logic notations of temporal PN of the above addressed temporal behaviors are (σ,Mc) |= ○t1, (σ,Mc) |= ◊t1, (σ,Mc) |= (¬t2 u t1), respectively.

Since temporal Petri nets follow linear-time temporal logic, the specification and verification of temporal properties is confined to a linear firing sequence. This suffers the same high complexity problem of other linear-time temporal logic pertaining to concurrence analysis (Sistla and Clarke 1985). We have extended the use of temporal PN in two perspectives— from classic PN to time PN (reachability tree to be exactly) and from linear-time temporal logic to branching-time temporal logic (Yang at el. 1998).

Definition *(Reachability Tree Logic):* RTL is a 5-tuple RTL = (P,T,M,A,R) where

1. P is a set of places, i.e., P=(p1,p2,...,pm);
2. T is a set of transitions, i.e., T=(t1,t2,...,tn);
3. M is a set of markings, i.e., M = (M_0,M_1,...,Mc,...,Mm). Mc = (pj), where pj ∈P. M_0 denotes the initial marking;
4. A is a finite set of arcs in a reachability tree denoting all possible binary relation between two markings, i.e., A ⊆ M$^{\chi}$M. An arc, Ak = tk(Mc, Mj), is a transition tk between two markings Mc and Mj, such that Mj is the resulting marking of firing tk at Mc, where tk∈T and Mc, Mj∈M;
5. R is a finite set of atomic propositions.

There are two atomic propositions in RTL: place propositions and transition propositions. A place proposition is denoted as the same notation of place pj. Mc |= pj if pj have at least one token at marking Mc; a transition proposition is denoted as the same notation of transition tj and Mc |= tj is true if tj is fireable at Mc. Every atomic proposition is a RTL state formula. We express a state formula f holds at a current marking Mc of a net N as N.Mc |= f, or simply Mc |= f if N is known. Mc |= f can be abbreviated as f when Mc is known. It is well known that the until operator u can be used to define eventuality operator ◊ as well as unless w operator, and eventuality operator can be used to define henceforth (always) operator □. Thus, assume that we have propositions f, f1, f2 and let a current marking be Mc. We only need two primitives operators: next operator ○ and until operator u, and two path quantifiers- there exist ∃ and for all ∀ to represent all RTL operators. For detailed definitions and theory of RTL. Please refer to (Yang at el.1998).

5.2 Analysis Methods

There are two fundamental techniques mostly used to analyzing behavioral properties. The first one is to use coverability (reachability) tree to enumerate all reachable markings. The other one is to utilize mathematical formalism of incidence matrix and state equation. As far as the analysis of temporal behavior is concerned, we propose a model-checking mechanism based on reachability tree of a Petri net.

5.2.1 The Coverability (Reachability) Tree

Through finding all reachable markings of a Petri net (N, M_0), we can conclude a tree representation called reacahbility tree. In the tree, each node represents a marking which is reachable from M_0 (the root). Each arc represents a transition firing, which transforms one marking to another. But if a Petri net was unbounded, the tree will become infinitely large. To make the tree finite, we introduce a symbol w which means “infinity”, where $w > n$, $w \pm n = w$ for each integer n. When the Petri net is unbounded, the tree is called as coverability tree; otherwise called reachability tree.

5.2.2 Incidence matrix and State Equation

For a Petri net N with m transition and n places, the incidence matrix of N, A = [ai,j] is a m×n matrix of integers in which the entry of i-th row j-th column in A is denoted as m×n matrix. Let A+ = [ai,j+], in which ai,j+ is the weight of arc from a transition ti to its output place pj, and let A- = [ai,j-], in which ai,j- is the weight of arc to a transition ti from its input place pj. Let ai,j = 0 if there is no arc connection between transition ti and place pj. We have that: ai,j = ai,j+ - ai,j- and A = A+ - A-. For state equation, let σ be a firing sequence leading from the initial marking M0 of N to a marking Md. $\Delta M = Md - M0$. X is a vector [x1, x2,…, xi,…, xm] in which xi denotes the firing counts of transition ti in σ. If $ATX = \Delta MT$ has non-negative integer solutions of X, then Md is said to be a reachable marking from M0. For example, in our NCUPN, the incidence matrix and the state equation of the Petri net in Figure 19 are shown in Figure 20.

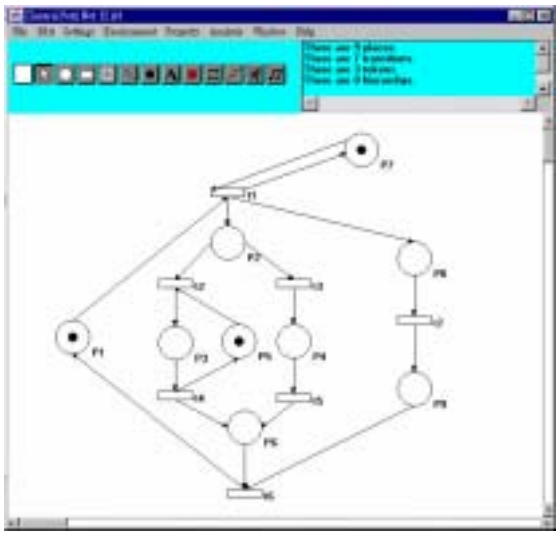

Figure 19. A Petri net PNa

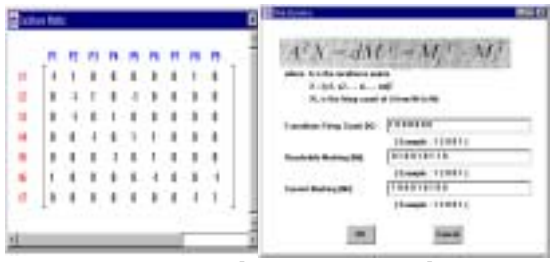

Figure 20. Incidence matrix and state equation

5.2.3 Model-Checking

Basically, a model-checking is a search mechanism used to check whether we can find a model to satisfy the to-be-checked formulas. Our model-checking mechanism (Yang et al. 1998) use the depth-first (left child first) traversal order starting from a current marking Mc, To check Mc |= (f1 u f2), for each firing sequence σi, σi ∈σ, the mechanism searches for the first marking Mk that satisfies f2 or the last Mk labeled with (f1 u f2) along the σi. If such Mk exists, then traverse the σi from Mc again to check whether all the markings Mj along the σi satisfy f1. Mj is labeled with (f1 u f2) if Mj satisfy f1, otherwise Mj is labeled with ¬(f1 u f2). If all of the Mj along the σi are labeled with (f1 u f2), then we conclude that Mc |= (f1 u f2) over σi. Otherwise we conclude that ¬Mc |= (f1 u f2) over σi. If we need to check a model for Mc |= ∀(f1 u f2), this mechanism will not stop traversal until we find the first σi, such that ¬Mc |= (f1 u f2) over the σi. If we need to check a model for Mc |= ∃(f1 u f2), the algorithm will not stop traversal until we find a σi such that Mc |= (f1 u f2) over the σi.

5.3 The Analysis of Classical Petri Nets

Base on our transformed classical Petri net, we shall represent an application for bank loan with a token, so in Figure 7 a token will be place in the starting place p1 for representing the initial state of the application. In this section, we focus on analysis techniques, which can be used to verify workflow properties. Once a system has been modeled, it exhibits two kinds of properties of the modeled system that is behavioral properties and structural properties. Behavioral properties are properties that dependent upon system's initial marking (state), whereas the structural properties do not. In (Aalst 1996), a 'good' structural characterization of workflow is to balance *AND/OR-split* and *AND/OR-join*. It is means that two parallel workflows initiated by an *AND-split* should not ended with an *OR-join*, two alternative workflows initiated by an *OR-split* should not ended with an *AND-join*. As a result, if we found that two alternative workflows initiated by an *OR-split* are ended with by an *AND-join*, then we know the *AND-join* will casue a deadlock. If two parallel workflows are initated by an *AND-split* and ended with an *OR-join*, then we know the two workflow processes are redundant. There are two fundamental techniques mostly used in Petri nets to analyzing behavioral properties. The first one is to use coverability (reachability) tree to enumerate all reachable markings. For our example of bank loan in Figure 7, the corresponding reachability tree is shown in Figure 21. The other one is to utilize mathematical formalism of incidence matrix and state equation. Our NCUPN provides both of the techniques. Once a wrokflow process is transferred using Petri nets, NCUPN can do the process property analysis automatically.

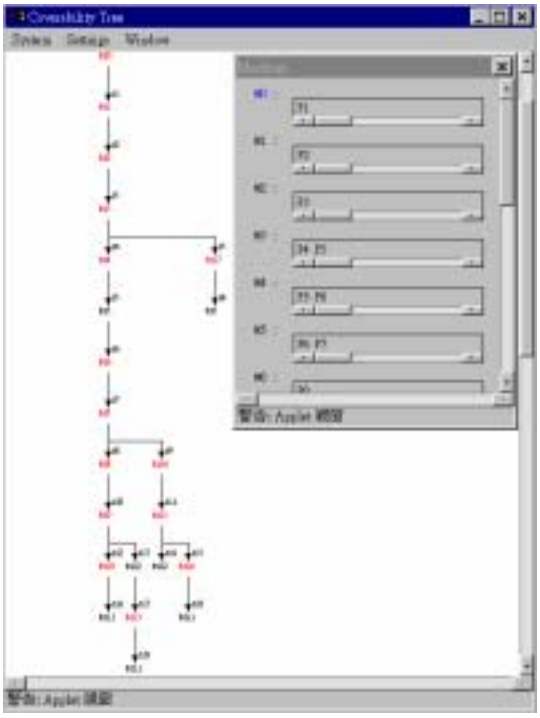

Figure 21. A reachability tree constructed from the Petri nets in Figure 7

We classify five situations under which workflow process will result in misbehaviors: co-exist of *AND-split* and *OR-join*; co-exit of *OR-split* and *AND-join*; deadlock; starvation; and safety. We refer to a resulting Petri net as the Petri net model of workflow process. If there is a co-exist of *AND-split* and *OR-join*, then the resulting Petri net will not be 1-bounded. For our example of bank loan in Figure 7, it is not a co-exist of *AND-split* and *OR-join*, and the resulting is 1-bounded as shown in Figure 22. If there is a co-exit of *OR-split* and *AND-join*, then the resulting Petri nets will be L0-live. For our example of bank loan in Figure 7, it is not a co-exit of *OR-split* and *AND-join*, and the resulting is L1-live as shown in Figure 23. If there is a deadlock, then the resulting Petri net will be L0-live. For our example of bank loan in Figure 7, it is not happen deadlock, because it is L1-live as shown in Figure 23. If there is a starvation, then the resulting Petri net is not B-fair, or for all synchronic distance of paired transitions is one. For our example of bank loan in Figure 7, it is not happen starvation, because it is B-fair as shown in Figure 24. The resulting Petri net is in safety condition if there are no such reachable unsafe markings. In the following, we will address those behavioral properties, which can be analyzed by using our NCUPN, and present how to use NCUPN to do the analysis.

Figure 22. Example in Figure 7 is boundness.

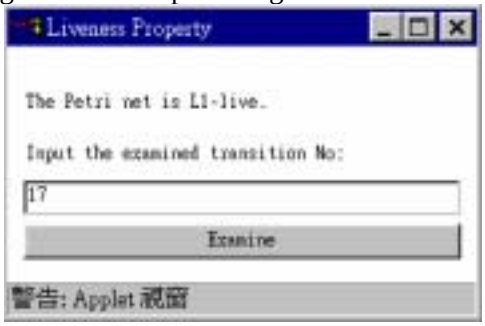

Figure 23. Example in Figure 7 is L1-Live.

Figure 24. Example in Figure 7 is B-fair.

5.4 The Analysis of Coloured Petri Nets

This section discusses how the Coloured Petri nets are analyzed. First, let's construct the reachability trees (i.e., occurrence graph) corresponding to the Coloured Petri nets (Jensen 1992). By means of the analysis method reachability tree, we can directly analyze some behavioral properties, such as reachability, boundedness, liveness, fairness etc on the tree. If the reachability tree can't satisfy those properties, we make modification directly on the class diagram and the statechart diagram in terms of Coloured Petri nets, and then transform the class diagram and the statechart diagram into the corresponding Coloured Petri net again. Analogously, we will also analyze those properties on the tree. This modification will continue until it can satisfy those properties. As the example of the bank loan process, after the applicant had filled out the application and delivered it to the clerk, the clerk double-checked all fields on the form and then took it to his supervisors for the approval. Those supervisors may serve in different departments so that for their signature the clerk may a step in their personal office and discussed with them. Each supervisor will review the related fields. Therefore, once one of the fields has errors or can't meet the regulations of the bank, the amount of the loan may be decreased or even rejected.

Based on (Jensen 1992) and readability, we decide to analyze the reachability and liveness properties. For example, when the bank had approved or rejected the application, we can know if such an application is approved or rejected by checking the reachable marking. In Figure 25 we construct the corresponding reachability tree of the Coloured Petri nets. The marking [p5 p10 p11] is the final marking of the reachability tree. According to it, we can check the value of its token is "true" or "false". The value "true" stands for that the application is accepted, while the value "false" stands for that the application is rejected. In other words, by means of the reachability property, we are capable of finding whether the marking had appeared in the reachability tree. Furthermore, the transition t2 commented with "evaluate application" is a hierarchical substitution (Jensen 1992). By its hierarchical structure, we can further specify a number of regulations in order to check the records of the applicant, such as credit, possessions, the amount of his loan etc. Analogously, the transition t5 commented with "evaluated(demand)" is a hierarchical substitution that can be further specified by other specific regulations. Nevertheless, we skip the specification and directly show our input and output. For more information, please see (Jensen 1992). The following test cases are provided for verifying our Coloured Petri net model. Then we enter this test case to verify the application the reachability tree of the Coloured Petri net as Follows in Table 3.

Table 3. A list with three test cases showing if the loans are accepted or rejected by checking their multiple conditions

	The number of the application	The amount the loan	Income	Credit	Possession	Grant
Input 1	1	1,000,000	50,000	50,000	100,000	Rejected
Input 2	2	2,000,000	10,000	500,000	1,000,000	Accepted
Input 3	3	50,000	20,000	50,000	100,000	Accepted

So far, we have completed the analysis of the reachability property. Similarly, for the liveness property we are able to verify L0, L1, L2, and L4-live state (Murata 1989) by using the above analysis method on the tree. For example, in Figure 10 let's specify number 3 application as the value of the token of the initial marking. That is, p3 have number 3 as the value of the token "d". In addition, we also put some tokens to form the initial marking of this Coloured Petri net. First, we construct its reachability tree, and then verify if t2 commented with "evaluateApplication" appears in the tree, i.e., check if an L1-live state appears in it. If not (i.e., L0-live), this implies that there is not number 3 application, and immediately by firing t3 another application will be chosen and dealt with. Of course, by verifying its liveness property, we can modify the CPN model until it can satisfy those desired behavior. As mention before, the modification is analogous to the analysis of the pervious reachability property.

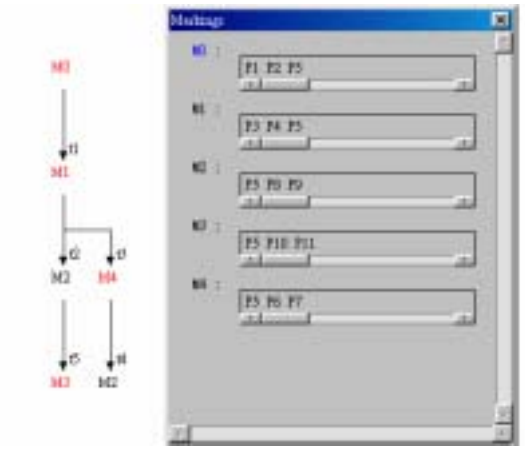

Figure 25. A reachability tree constructed from the Petri nets in Figure 11

6. Conclusions and Future Research

In this paper, we have presented an UML approach to model business processes. We adopt use case diagram to capture the requirements of business processes, class diagram to exhibit information structure of business processes, activity diagram to express logical steps of business processes. All these UML modeling comply with the standard six workflow primitives defined by WFMC. We will continue to develop workflow management system and apply it in electronic commence in our future research.

Acknowledgements

This research is supported by the National Science Council in Taiwan under grant NSC 89-2213-E-008-009.

References

1. Aalst, W. M. P. "Three Good Reasons for Using a Petri-net-based Workflow Management System," *Proc. Of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96). Eds. S. Navathe and T. Wakayama, Camebridge, Massachusetts,* Nov. 1996, pp. 179-201.
2. Booch, G., Rumbaugh, J., and Jacobson, I. *The Unified Modeling Language User Guide*, Addison-Wesley Longman, Inc., 1998.
3. Brooks, F. P. "No Silver Bullet: Essence And Accidents of Software Engineering," *Information Processing '86*, 1986, pp. 10-19.
4. Lawrence, P. *Workflow Handbook 1997*, Wiley and Sons Ltd, New York, 1997.
5. Lei, Y., and Singh, M. P. "A Comparison of Workflow Metamodels," 1997, http://osm7.cs.byu.edu/ER97/workshop4/ls.html.
6. Jensen, K. *Coloured Petri nets: Basic concepts, Analysis methods, and Practical use, Vol. 1-3*, Springer-Verlag, 1992.
7. Muller, P. A. *Instant UML*, Wrox Press Ltd., 1997.
8. Murata, T. "Petri Nets: Properties, Analysis and Application," *proceeding of IEEE*, Vol. 77, No. 4, 1989, pp. 541-580.
9. Peterson, J. L. *Petri net theory and the modeling of systems*, Prentice-hall, Central Book Company, Taipei, Taiwan, 1981.
10. Rational, and UML partners, "UML Notation Guide version 1.1," 1997, http://www.rational.com/uml/resources/documentation/notation/index.html.
11. Sistla, A.P., and Clarke, E. M. "Complexity of Propositional Linear Temporal Logics," *Journal of ACM*, Vol. 32, No. 3, July 1985, pp. 733-749.
12. Suzuki, I., and Lu, H. "Temporal Petri Nets and Their Application to Modeling and Analysis of a HandShake Daisy Chain Arbiter," *IEEE Trans. on Computer*, Vol. 38, No. 5, May 1989, pp. 696-704.
13. Veijalanen, J., Lehtola, A., and Pihlajamaa, O. "Research Issues in Workflow Systems," October 1995.
14. Workflow Management Coalition, *Workflow Management Coalition The Workflow Reference Model*, Nov 1994.
15. Workflow Management Coalition, *Workflow Management Coalition Terminology & Glossary*, June 1996.
16. Yang, S. J. H., Chu, W., Lin, S., and Lee, J. "Specifying and Verifying Temporal Behavior of High Assurance Systems Using Reachability Tree Logic," *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*, 1998, pp. 150-156.

About the Authors

Stephen J.H. Yang (楊鎮華) is an associate professor in the Department of Computer and Communication Engineering at the National Kaohsiung First University of Science and Technology. Dr. Yang was the Founder and CEO of T5 Corp. Taiwan. From 1996 to 2000, he was an associate professor in the Department of Computer Science and

Information Engineering at the National Central University, Taiwan. From 1995 to 1996, he was an associate professor in the Department of Information Engineering at the Feng Chia University, Taiwan. Dr. Yang is the coauthor of the book *Distributed Real-Time Systems: Monitoring, Visualization, Debugging, and Analysis* (John Wiley & Sons, Inc., 1996) and the coeditor of the book *Monitoring and Debugging of Distributed Real-Time Systems* (IEEE Computer Society Press, 1995). His research interests include Petri nets, software engineering, e-Business applications, software component, and object-oriented technologies.

Dr. Yang received a B.E. in Computer Science from the Tamkang University, Taiwan in 1985 and an M.S. and a Ph.D. in Electrical Engineering and Computer Science from the University of Illinois at Chicago in 1993 and 1995, respectively. He is a member of IEEE Computer Society and ACM.

Chyun-Chyi Chen (陳群奇) received his B.S. degree in Mathematics from Fu Jen Catholic University, Taipei, Taiwan, in 1995; and his MS degree in Industrial Engineering from Chung Yuan Christian University, Chung-Li, Taiwan, in 1997. He is currently a Ph.D. student in the Department of Computer Science and Information Engineering at the National Central University, Chung-Li, Taiwan.

In 1997, he joined the Laboratory of Software Engineering, National Center University, where he is working on the technologies of software engineering, workflow management system, Petri nets and temporal logic. His current interests include workflow management system, object-oriented technologies, Petri nets, temporal logic, rule-based system, enterprise resource planning systems, supply chain management system, and electric commerce.