# Edinburgh Research Explorer

# Knowledge management using business process modeling and workflow techniques

**Citation for published version:**
Kuo, H-L, Chen-Burger, Y-H & Robertson, D 2003, Knowledge management using business process modeling and workflow techniques. in *18th International Joint conference on Artificial Intelligence (IJCAI), Workshop on Knowledge Management and Organizational Memories.*

**Link:**

**Document Version:**
Peer reviewed version

**Published In:**
18th International Joint conference on Artificial Intelligence (IJCAI), Workshop on Knowledge Management and Organizational Memories

# Knowledge Management using Business Process Modeling and Workflow Techniques

**Hsiang-Ling Kuo[1], Yun-Heh Chen-Burger[2], Dave Robertson[3]**
[1] School of Informatics, The University of Edinburgh, UK
email: s0125762@ed.ac.uk
[2] AIAI, The University of Edinburgh, UK email: jessicac@inf.ed.ac.uk
[3] CISA, The University of Edinburgh, UK email: dr@inf.ed.ac.uk

## Abstract

Enterprise Modeling (EM) methods are recognised for their value in providing a more organised way to describe a complex, informal domain. A problem with EM is that it does not always provide direct input for software system development. There is a "gap" between EM and software systems. One way of bridging this gap is to provide a formalisation, here called that subsumes a wide variety of core modeling notations in a single using a business process language. It is possible to have different views of what is core to such a language but our attempt at such a view is articulated in the Fundamental Business Process Modeling Language (FBPML), which is a merger of IDEF3 and PSL. A workflow language, the FBPML Workflow Language (FWFL), is constructed and used to provide a declarative description of a workflow system. FWFL is tested in the "PC-configuration" domain. We also suggest using a validation and verification support framework to analyse and verify the business process model (BPM). Finally, some complexity results are presented for this type of modeling.

## 1 Introduction

Business are becoming larger and more diverse, thus operations are more complex than ever. Although information technology is widely applied in business operations, it still lacks a precise means of communication between business model and software system development. EM methods are well recognised for their value in providing an organised way of describing a complex, informal domain, and are often used as a tool for Knowledge Management (KM). There are many types of EM methods such as business modeling methods, process modeling methods, organisational modeling methods and related ontology design methods. However, EM does not always provide direct input for software system development which leaves a gap between enterprise modeling and software systems.

This paper tries to bridge the gap between *Enterprise Modeling* (EM) and *Software Systems*. To approach this, we have created a formalisation called "FWFL" for all models using FBPML which provides a declarative description of a workflow system. A workflow system may then be designed and implemented based on "FBPML" and "FWFL". This would play an important communication role in the operation of an organisation.

In order to verify and analyse the BPM, a three-level framework is also introduced as a means of analysing BPMs and workflow systems. Finally, the complexity of BPMs and some comparisons with other related work are discussed.

## 2 Literature Review

Many business process modeling languages have been invented for different business process modeling needs. In this section, we will review two types of process modeling language and introduce a third one – Fundamental Business Process Modeling Language (FBPML).

### 2.1 IDEF3 Process Description Capture Method

IDEF3, a process description capture method, has two aspects: capturing process flow and object state. The primary goal of IDEF3 is to provide a well-structured method by which a domain expert can express knowledge about the operation of a particular system or organisation [Mayer *et al.*, 1995]. It also captures the behavior of an existing or proposed system by structured description. Because of its well-structured approach, we can use IDEF3 as a knowledge acquisition device for describing what a system does or how an organisation works.

IDEF3 includes two forms of description: a *process flow description* and an *object state transition network*. A process flow description describes "how things work" in an organisation [Mayer *et al.*, 1995]. It focuses on the processes and their temporal, causal, and logical relations. An object state transition network focuses on objects and their state change behaviors. This paper focuses on the process flow description because it is related to what we use in this paper.

An IDEF3 process flow description captures a description of processes and the relationships between them. It provides a graphical and structural representation that domain experts and analysts from different disciplines can use to communicate with each other. This includes knowledge about events and objects involved in the process, and the constraining relations which determine the behavior of each occurrence (process and object). It uses *UOB (units of behavior), links, junc-*

*tions*, *referents* and *notes* to represent the processes and their relationships (such as temporal ordering).

## 2.2 Process Specification Language

PSL stands for Process Specification Language. It is an interchange language that allows applications to exchange discrete process data [Schlenoff *et al.*, 1997]. It provides a common language between different applications and captures the necessary process information from any given application. The goal of PSL is to facilitate communication between those applications by using PSL-based translators. For example, suppose there are *n* different applications, that will communicate with each other. If there is no intermediate language like PSL, it requires $O(n^2)$ translators for them to communicate. But with PSL language as a standardized communication medium, the number will reduce to $O(n)$.

PSL provides formal specification for semantics in process models due to lacking or inadequate specification in existing approaches. With PSL, process information can be exchanged between a variety of applications. This formal specification is the PSL *"ontology"* as it focuses not only on the terms of the ontology but also their meanings.

The PSL ontology has three notions: *language*, *model theory* and *proof theory*. A *language* is a lexicon (a set of symbols) and a grammar (a specification of how these symbols can be combined to make well-formed formulas). The lexicon in PSL is a set of logical symbols (e.g. boolean, quantifiers) and nonlogical symbols (i.e. PSL expressions, such as constants, functions, symbols, and predicates)[Schlenoff *et al.*, 2000]. In *model theory*, PSL provides a mathematical characterization of the semantics, or meaning, of the language of PSL [Schlenoff *et al.*, 2000]. *Proof theory* consists of three components: *PSL core*, *foundational theories*, and *PSL extensions*.

The *PSL core* is a set of axioms written in the basic language of PSL. These axioms provide a syntactic representation and semantic description of the PSL model theory [Schlenoff *et al.*, 2000].

A *foundational theory* has sufficient expressive power for giving precise definitions of, or axiomatizations for, the primitive concepts of PSL [Schlenoff *et al.*, 2000].

*PSL extensions* are expressions that are not included in the PSL core. It provides extra usage for expressing more complicated processes.

## 2.3 Fundamental Business Process Modeling Language (FBPML)

FBPML, a visual modeling language, merges IDEF3 and PSL. This language is designed to support both software and workflow system development. It offers precise semantics and can express business processes in logical sentences.

There are three types of nodes: **Main Node**, **Junction** and **Annotation**. **Main Nodes** include *Activity*, *Primitive Activity*, *Role* and *Time Point*. Process is the main concept of process modeling languages. In FBPML, as in PSL, an activity is used to represent a process. In this document, we will use *process* and *activity* interchangeably.

*Activity* and *Primitive Activity*: An *Activity* describes a type of process that may be decomposed into sub-processes. This is called *"decomposition"*. When all sub-processes are finished, the high level process is also finished. *Primitive Activity* is a leaf node activity that may not be further decomposed. However, there may be alternative ways of executing a process. When one alterative process is not executed and finished properly, another alternative process may collaborate with the current one to accomplish the task. We call these sub-alternative processes *"specialisation"*. In FBPML, three main components of an activity are *trigger(s)*, *precondition(s)*, and *action(s)*. An activity also has an unique *hierarchical position(HP)* and a *name* to identify it.

*Time* and *Role*: The definition of *Role* in FBPML is useful, an enabler may play a Role that includes a set of activities and may have responsibilities for these activities. *Time Point* indicates a particular point in time during the process model.

**Junctions** are widely used in many process modeling languages. In FBPML, there are four different types of junction: *Start*, *Finish*, *And* and *Or*. The *Start* and *Finish* junctions represent the beginning and end of a BPM. *Start* is the entry of a BPM. *Finish* is the point at which the model stops.

*And* or *Or* junctions indicate a one-to-many relationship, and a temporal constraint between the activities connecting to them [Chen-Burger *et al.*, 2002]. Both of these junctions have two kinds of interpretation: *joint* and *split*. They represent the different topologies of a BPM. Figure 1 shows these four different types of topology.
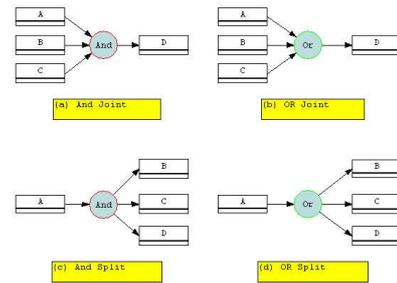


Figure 1: And Joint, Or Joint, And Split and Or Split Topology

**And_Joint** or **Or_Joint** indicates that there is more than one process preceding the "And" or "Or" junction but there is only one process following the junction. 1(a) and (b) show these kinds of junction. Both have three in-coming processes (A,B,C) and one out-going process (D). **And_Joint** indicates the process execution sequence and the temporal constraint on the process. It means that all the processes (A,B,C) on the left-hand side must be finished before process D can be started. If one of the left-hand side processes cannot be finished, the entire flow cannot continue to the next stage. **Or_Joint** means that when at least one of the left-hand side processes is finished then process D can be started; it does not need to wait for the other processes to be finished.

**And_Split** or **Or_Split** means that only one process will proceed to the "And" or "OR" junction, but more than one process will follow the junction. 1(c) and (d) illustrate this.

**And_Split** indicates that as long as the preceding process is finished then all the following processes become temporally qualified and may also be started. It also indicates that all the following processes must be triggered, but are allowed to be finished at some later time. **Or_Split** means at least one of following processes will be triggered and executed properly, after the preceding process is finished. It does not have any constraint about how many processes will be triggered. It may be one, two or more, depending on the trigger conditions of those processes. The execution sequence of triggered processes may be parallel or sequential.

**Combination of "And" and "Or" Junctions:** The "OR" and "And" junctions may also be combined to represent a more complicated BPM. There are four different kinds of combination. Suppose process A is connected with processes B,C, and D. Process E follows processes B,C and D in these different combination. The first type of combination is "And_Split" (figure 1 (c)) and "And_Joint" (figure 1 (a)). It means that when process A is finished then process B,C and D will be and must be started. After all the processes B,C and D are finished [1], then process E can start. It has the strictest restriction in a BPM. The combination of "Or_Split" (figure 1 (d)) and "Or_Joint" (figure 1 (b)) means that after process A is finished, *at least one* of the processes B, C and D will be started and executed. Process E will not be started unless one of the triggered processes is finished. It is a looser constraint than an "And_Split" and "And_Joint" junction.

The combination of "And_Split" (figure 1 (c)) and "Or_Joint"(figure 1 (b)) means that when process A is finished, all the processes B,C and D must be started and executed. If process B, C or D is finished, then process E can be started. It is different from an "And-And" junction in that the "Or_Split" (figure 1 (d)) and "And_Joint" (figure 1 (a)) junction indicates that *at least one* of the processes B,C and D will be started and executed after process A is finished. Process E will not be started unless "*all of the triggered processes*" [2] are finished. The triggered processes may be a combination of some of them. Because of the "Or_Split" junction, it does not need to trigger all the preceding processes. It thus has more flexibility than the "And_Split" junction.

**Annotations** include *Idea Note* and *Navigation Note*. *Idea Note* records information which is related to the processes but not part of the process model. *Navigation Note* records the relationships between diagrams in a model [Chen-Burger *et al.*, 2002]. Neither of them contribute to the formal semantics of the process model. Instead, they are used to help users to understand the processes more clearly from an intuitive point of view.

Nodes (Main Nodes and Junctions) are connected by **links**. Two types of links are provided: the *precedence-link* and the *synchronisation-bar*. A *precedence-link* indicates a temporal constraint between two processes. It means that activity B cannot start until activity A has finished. A *synchronisation-bar* also places a temporal constraint between two time

---

[1]There is no restriction about the execution sequence of these three processes.

[2]These indicate that the pre-processes of the "And_Joint" junction which are triggered.

| ITEM | IDEF3 | PSL | FBPML |
|---|---|---|---|
| **Property** | Process modeling language | Interchange language between different manufacturing applications | Business modeling language especially supports software and workflow system development |
| **Notation** | Rich Graphic Notation | Ontology and Formal Semantics | Simpler version Notation but semantics are presented |
| **Basic Process Description** | UOB (Unit of Behavior) | Activity, Primitive-Activity | Activity, Primitive-Activity |
| **Distinguish terms between process, activity and task** | √ | √ | × |
| **Link between processes** | Precedence Links with Different types of Constraints | Ordering Relation over Activities(ext) | Precedence-Link, Synchronisation-Bar |
| **Junction** | AND, OR, XOR | AND, OR, XOR Junction(core) Junction(ext) | √ (not the same as PSL and IDEF3 but is an extension and refinement of both) |
| **Time** | Not in the notation but may be expressed informally | Duration(ext), Temporal Ordering Relation(ext) | Time point, duration, length |
| **Role** | × | × | √ |
| **Annotation** | Referent and Note | × | Idea Note and Navigation Note |
| **Decomposition** | A process may be decomposed into sub-processes | SubActivity(ext) | A process may be decomposed into sub-processes |
| **Specialisation** | √ | × | √ |
| **Execution Logic** | × | × | √ |

Table 1: Comparison Between IDEF3, PSL and FBPML

√ – Yes
× – No/Not applied
core – PSL core concept
ext – PSL extension concept

points. This notation enables any time points to be made equivalent and therefore enables process operations to be synchronised.

## 2.4 Comparison between IDEF3, PSL and FBPML

After briefly reviewing the IDEF3, PSL and FBPML, we will focus on their similarities and differences. Table 1 summaries this brief comparison.

**Similarities:** IDEF3, PSL and FBPML all focus on process expression techniques to provide a standard methodology when describing a process. These three business process representations allow domain experts to express knowledge about how a system works, and can provide a common language between different applications.

**Differences:** Although IDEF3, PSL and FBPML are conceptually similar, in table 1 we lists the major differences between them. IDEF3 provides a graphical way to represent

processes, allowing the logic of processes to be more easily represented. Although IDEF3 has this advantage; it lacks an unambiguous semantic description for its notation. By contrast, PSL has a well-defined ontology and formal semantics but lacks graphical notations. Users cannot easily define processes in this language without proper training in logical languages. FBPML combines aspects of both IDEF3 and PSL to obtain the advantages of their different aspects. It provides a simpler and more pragmatic modeling language suitable for workflow system design.

## 3 Devising a logic-based Workflow Language for FBPML

FBPML is a visual and conceptual language. It captures and describes the business processes of an organisation. Besides describing a model, it also allows BPMs to be analysed, redesigned and checked. Through FBPML, the tasks and operation of an organisation can be more easily understood. In order to provide the properties described above, FBPML has a declarative reading (understood independently of any particular computational procedure) as well as an operational reading when combined with a particular computational procedure. These are exactly the principles that FBPML embodies.

While FBPML gives precise execution logic for processes, it allows multiple versions of implementation of the workflow engine. This is because the FBPML specifies how processes should be enacted but does not specify the workflow engine that enacts those processes. In this section, we have constructed a workflow language *"FWFL" (the FBPML WorkFlow Language)* based on FBPML. In section 5, one version of the workflow engine will be introduced based on *"FWFL"*. The definitions of FWFL will be introduced in the following section.

### 3.1 FBPML WorkFlow Language (FWFL) Design

**Process:** The predicate defines a process. It has six parameters:*ProcessId*, *ProcessName*, *Pstate*, *TrigCond*, *PreCond* and *Action*.

$$process(ProcessId, ProcessName, Pstate, TrigCond, PreCond, Action)$$

*ProcessId* defines the ID of a process. It must be unique to all processes in the BPM. *ProcessName* describes the name of a process. It does not have any impact on the execution, but is used only for human interpretation. It's purpose is to help the user understand what the process is. *Pstate* records the status of a process. There are two types of status in a process – "Triggered" and "Completed". "Triggered" means that the process is already triggered and is temporally qualified to be executed. "Completed" means that the process has already been executed properly, i.e. it is already finished. *TrigCond* defines the trigger condition(s) for the process. It is composed of a *triggered event* or values about a state, e.g. *attributes* and their values of an entity. *PreCond* defines the pre-condition(s) for the process. It is composed of *attributes* and their values of an entity that a process manipulates[3]. A simple example

---

[3]It also has a condition – *delay_time* which describes a delay condition.

will be used to illustrate this later. A user can understand what the process looks like simply by reading through its description. Example 1 shows that process "a" needs an event to trigger the process. The content of this trigger event can be read simply through the description.

*Actions For Processes:* In FWFL, eight different types of action are provided. They are *create_entity*, *get_newValue_from_usr*, *get_uptValue_from_usr*, *add_attribute*, *delete_attribute*, *update_attribute*, *refer_attribute_of_entity* and *delete_entity*.

The action *create_entity* creates a new entity and adds it into the entity database. To carry out this action – the attribute and its value for that entity are needed at execution time. The action *get_newValue_from_usr* gets a value from the user at run time, as a workflow system may have interactions with other systems as well as with the user. This action provides an interface through which a workflow system can obtain new data from a user. The action *get_uptValue_from_usr* is similar to get_newValue_from_usr. The only difference is that it gets an updated value to update an existing attribute. The actions *add_attribute*, *delete_attribute* or *update_attribute* add, delete or update attributes in an existing entity. The action *refer_attribute_of_entity* refers to the value of an attribute provided by another entity. The action *delete_entity* deletes an existing entity. Sometimes the actions being carried out by one process may conflict with another process. The workflow system needs to deal with this situation and provide appropriate warning messages.

```
Example 1:
 process(a, receiveCustomerReq, Pstate,
        [exist(event_occ(EventId,
          custom_req_for_pc_spec,
          created,
          attribute(Attr)))],
        [true],
        [create_entity(attribute(Attr))]).
```

**Instance:** A process instance is an actual running process at execution time. A process (type) may have more than one instance depending on the number of events. The definition of an instance is inherited from process, such as TrigCond, PreCond and Action. The only difference between a process and an instance is that process' variables *ProcessId*, *ProcessName* and *Pstate* are used instead of *InstanceId*, *InstanceName* and *Istate*, but the definitions of these parameters are the same. In addition, an instance has a parameter – *BeginT/EndT* that is different from process. It records the start and end time of that instance. As we know, an instance is the actual executed process, therefore, its variables are instantiated at run time. The workflow system needs to record this information and carry out some checking at execution time. A predicate–"instance" is therefore:

$$instance(InstanceId, InstanceName, Istate, TrigCond, PreCond, Action, BeginT/EndT)$$

**Attribute:** Attributes are used everywhere in FWFL such as "TrigCond", "PreCond", "Action", "entity" and "event". Attributes are defined in predicate:

$$attribute(EntityName - AttributeName/AttributeValue)$$

This means that the attribute, *AttributeName* belongs to a particular entity, *EntityName*. Its value will be assigned at execution time. For example, "customer-name/NameV" means the entity "customer" has an attribute "name". The value has not been assigned yet.

FWFL also allows two other different kinds of attributes to be used. These two attribute definitions are slightly different from the previous one. The first type is the *multiple-value* attribute. For example, "ioBoard-capability/(normal-graphics-long)" means that the entity "ioBoard" has an attribute "capability", and "capability" has multiple values which are "normal", "graphics" and "long". These three values appear at the same time in the attribute "capability". The second type is the *alternative-value* attribute. For example, "box-color/[white,silver,black]" means that the entity "box" has an attribute "color". The values of color are only one of "white", "silver" or "black".

**Attribute Domain and Attribute Value:** The domain of an *attribute* is defined by

$$att\_domain(Class, AttributeName, Domain)$$

The attribute, *AttributeName*, indicates that "AttributeName" is an attribute for instances in a class, "Class". The field *Domain* defines the domain of values for the attribute. For instance, $att\_domain(processor, type, [t1, t2, t3, t4])$ means that Class "processor" has an AttributeName "type", and its domain is "t1","t2","t3" or "t4". *Attribute Value* is defined by

$$att\_value(Class, AttributeName, Value)$$

It means "AttributeName" belongs to class "Class". The value of this AttributeName is "Value". For example, $att\_value(customer, orderNo, value(1))$ means that Class "customer" has an AttributeName "orderNo', and its value is "1".

**Entity:** An entity represents a class in the world. It defines the properties of an entity.
The predicate is:

$$entity(EntityName, EntityId, EntityState, EntityAttribute)$$

The *EntityName* is the name of the entity. The *EntityId* is the ID of the entity. The same entity type will have the same entity name but different IDs. Thus, the ID is unique. The *EntityState* records the state of the entity. There are two types of the state: "valid" and "invalid". The entity is "valid" when it is created; it is "invalid" when it is deleted or after it reaches a particular time point. The *EntityAttribute* contains the attributes and their values for that entity. At execution time, every occurrence of this entity must be bound by these properties. An entity occurrence is defined as

$$entity\_occ(EntityName, EntityId, EntityState, EntityAttribute)$$

which is inherited from the entity definition.

**Event:** An event defines the properties of a triggered event. It is represented as

$$event(EventId, EventType, EventFlag, EventAttribute, Time)$$

The *EventId* indicates the ID of an event. The *EventType* is the type of an event. The *EventFlag* records the state of an event [4]. The *EventAttribute* contains the attributes and their values for the event. *Time* records the trigger time for this event. The event predicate defines the properties that are needed in an event. In the same way, the event data is represented as an event occurrence. An event may have different occurrences. We use

$$event\_occ(EventId, EventType, EventFlag, EventAttribute, Time)$$

---

[4]Two flags – "new" and "created" are defined in FWFL.

to express it.

**Junctions and Models:** The predicate

$$junc(ModelId, JunctionType, PreProcesses, PostProcesses)$$

is used to represent a junction. For example, if we want to represent a junction *Or_Split* which is connected with "b" and "c,d,e,f" for Model "m1", we can represented is as $junc(m1, or\_split, [b], [c, d, e, f])$

## 4 Validation and Verification Support Framework for Business Process Modeling

In this research, a three-level framework is provided to analyse the BPM. The framework includes "*model behavior*", "*detailed model testing*" and "*instantiation of business scenario – case study.*" This framework is intended to give users a more systematic structure when analysing a BPM. The framework first addresses the process flow control issues and then focuses on the detailed process analysis. The analysis of this framework may also be divided into two categories: *syntactic* and *semantic* [Sadiq and Orlowska, 1996]. The invalid use of a business process modeling language results in *syntactic errors*. *Semantic errors* occur due to incorrect modeling of business processes or getting into erroneous situations because of unanticipated combinations of task execution [Sadiq and Orlowska, 1996]. Based on these two types of error, we define our two types of critique as *syntactic critique* and *semantic critique*. We verify them and give advice in the three-level framework.

Figure 2 demonstrates our three-level framework and shows the relationships between level 1, 2 and 3. Level 1 considers the overall model behavior to find the appropriate topology for the BPM and carries out the syntactic critiques. Level 2 captures the topology features from level 1, carries out the semantic critiques and eliminates impossible execution sequences. Level 3 executes the BPM using business scenarios (entity data) in a particular domain and attempts to validate the model. Because the examined problem space for possible execution paths is reduced by each level as more information is presented by the model, we present our three-level approach in a three-layered oval graph. The detailed descriptions will be introduced in the following sections.
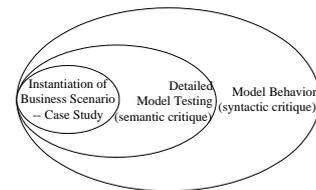


Figure 2: Overall Framework for Business Process Modeling

### 4.1 Model Behavior Level

*Syntactic critiques* are checked at the *model behavior* level. The types of syntactic critiques are shown in tables 2 and 3. This includes two parts– *syntactic errors* and *syntactic warnings*. "Model behavior" is also examined at this level. The *model behavior* indicates all the possible actions that the

| Error Types | Type of junction | Error and explanation |
|---|---|---|
| Junction logical error | And_Joint | more than one outgoing node |
| | Or_Joint | more than one outgoing node |
| | And_Split | only one outgoing node |
| | Or_Split | only one outgoing node |
| | Start & Link | more than one outgoing node |

Table 2: Syntactic Errors

| Warning Types | Explanation |
|---|---|
| Connecting warning | The link does not connect properly |
| Redundancy warning | More than one of the same node connected together |

Table 3: Syntactic Warnings

model may carry out. The first level in this framework generates this kind of model behavior. At this level, all the possible triggered results and the resulting execution sequences are determined. The users may know all of possible behaviors of the BPM using this facility. This facility is useful in understanding the behavior of a similarly complicated model. Through this simple behavior enumeration, the user may have a rough idea about what kinds of flow may be executed. Particularly, the user may know the state enumeration and all possible execution sequences. Although the BPM that we deal with is written in FBPML, this approach is generic to other business process modeling languages.

### 4.2 Detailed Model Testing Level

The main purpose of level 2 – *detailed model testing* – is to provide *semantic critiques* for the BPM such as *reachability analysis*[5], *potential deadlocks* and *irrelevant nodes*. As for syntactic critiques, it may be divided into *semantic errors* and *semantic warnings*. Table 4 and 5 summaries these critiques.

This detailed testing mechanism considers the logical meaning of the junctions, preconditions and actions of the processes together to verify the semantics of a BPM. It also figures out the possible execution results of the BPM.

At this level, not only the logical meaning of the junctions, but also the detailed preconditions and actions of the processes are considered. Neither level 1 nor level 2 consider trigger conditions, because all possible triggered results are

---
[5]"Reachability analysis" is used to describe the construction of a state-transition model of a system from models of individual processes [Yeh and Young, 1991]

| Error Types | Explanation |
|---|---|
| Unreachability error | The precondition of the process can never be satisfied, and the process will never be executed. |
| Deadlock error | A process(A) waits for another process(B) and process(B) waits for process(A) at the same time. |
| Termination problem | Determining whether a workflow structure can reach a terminating state [ter Hofstede *et al.*, 1996]. |

Table 4: Semantic Errors

| Warning Types | Explanation |
|---|---|
| Irrelevance warning | A process that does not use outputs from any other processes and does not produce inputs for other processes to use. |

Table 5: Semantic Warnings

listed at these two levels. The analysis of level 2 is based on the results from level 1. The analysis of level 1 enumerates all possible results of the BPM. Some detailed checking are added at level 2 to eliminate all impossible execution sequences while keeping all possible ones. A termination problem may also be detected at this level.

### 4.3 Instantiation of Business Scenario Level

At level 3, *instantiation of business scenario – case study*, the entity data is instantiated to a BPM. At this level, a user scenario adapted from AKT project[6] in the "PC configuration" domain has been used to test these ideas. The BPM is shown in figure 3. It is different from level 1 and level 2 in that level 1 and 2 focus on overall business process model simulation, whereas level 3 focuses on a special case study. At level 3, a workflow system is executed and may create instances of processes at run time which depend on the given input data. The final flow is determined due to the attributes of this data. At this level, conflicting actions are also checked and are signaled by warning messages.

A workflow system directly mapped to FWFL is implemented at this level. A BPM described in FBPML is used at this level, and may be checked and run in this workflow system. The detailed system architecture and implementation are discussed in section 5. The three-level framework becomes complete because model behavior is tested and some model checking is provided in level 1 and 2, and a case is used to analyse the BPM at level 3.
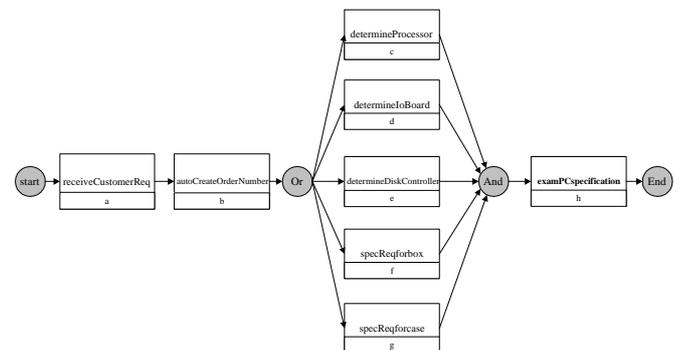


Figure 3: BPM adapted from AKT Project

---
[6]Advanced Knowledge Technologies (AKT) Project is an IRC (Interdisciplined Research Council) project. This project is used to develop and integrate relevant AI techniques in the larger process of knowledge management. Various public web sites are available at: http://www.aktors.org/

**Case Study**

The case used in this research is adapted from the AKT project [Chen-Burger, 2002a] in the "PC configuration" domain as shown in figure 3. A BPM and an entity database are needed for this workflow engine[7]. The BPM is composed of a *Process Specification*, a *Junction Specification* and an *Entity Specification*. Definitions of processes are stored in the *Process Specification*. It is written in FWFL which provides direct input to the workflow engine. If a BPM is changed, the definitions of the processes written in FWFL change correspondingly. The purpose of *Junction Specification* is to record the logical connections of the BPM. *Entity Specification* stores the definitions of the entities and events (i.e. the event occurrences and entity occurrences are stored in the entity database). Because the user needs to determine how many steps that the flow needs to run, the flow can stop at a step which is stipulated by the user.

Example 1 in section 3.1 illustrates a part of the *Process Specification*. *Junction Specification* is shown in example 2.

```
Example 2:
junc(m1,start,[],[a]).
junc(m1,link,[a],[b]).
junc(m1,or_split,[b],[c,d,e,f,g]).
junc(m1,and_joint,[c,d,e,f,g],[h]).
junc(m1,link,[g],[h]).
junc(m1,end,[h],[]).
```

Example 3 illustrates a part of the *Entity Specification*.

```
Example 3:
event_occ(e1,
  custom_req_for_pc_spec,
  new,
  attribute([entity-id/'e1',
            customer-name/'John',
            customer-doB/'13-06-70',
            customer-gender/male,
            customer-tel/'0131-5563432',
  spec([box-color/white,ioBoard-length/short,
      ioBoard-capability/(fast-_A-_B)])]),1).
  entity_occ(ioBoard,io1,valid,attribute(
      [ioBoard-type/io1,ioBoard-slot/3,
      ioBoard-length/short,
      ioBoard-capability/(fast-graphics-short)]))).
```

The BPM in example 3 describes a process flow for "PC-configuration for the customer" as shown in figure 3. The trigger conditions of process c, d, and e are "true" (i.e. they may be triggered automatically), so they must be triggered when the flow is running. The user can easily understand the process because a PC must consist of these three parts. Processes "f" and "g" are different. They are triggered only if the customer has a special requirement. The *Or_Split* junction is used here. It indicates that not all following processes need be triggered. The *And_Joint* is used in the next junction. It indicates that all the triggered processes must be finished at some time (i.e. the workflow must find a solution for the customer). The testing result about a case is shown in example 4.

```
Example 4:
 Model: Model1
 customer: (Mary, John)
```

Special Requirements : John's special requirement is satisfied, Mary's is not satisfied.

John's special requirement is:

```
sepc([box-color/white,ioBoard-length/short,
     ioBoard-capability/(fast-_-_)]]
```

---

[7]We construct a BPM here, but FBPML may also be used to construct a manufactory process model.
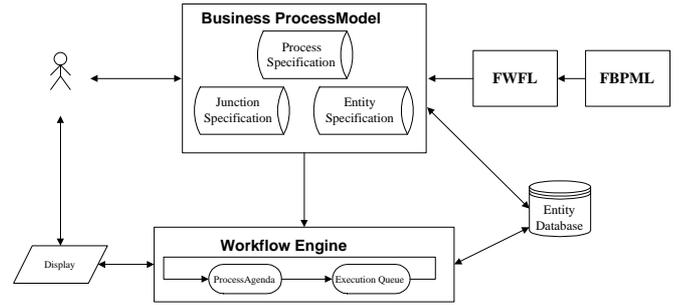


Figure 4: System Architecture of the FWFL Workflow Engine

Mary's special requirement is:

```
sepc([case-type/c1,box-color/black])
```

The Execution Sequence:

```
a1-i-(e1-John),a1-i-(e2-Mary),b1-i-(e1-John),b1-i-(e2-Mary),
f1-i-(e1-John),e1-i-(e1-John),d1-i-(e1-John),c1-i-(e1-John),
g1-i-(e2-Mary),e1-i-(e2-Mary),d1-i-(e2-Mary),c1-i-(e2-Mary),
h1-i-(e1-John)
```

The Execution Result:

```
The process f1-i-e2-Mary cannot be executed.
The reason may be the precondition cannot be
satisfied or the action has errors!
The requirement of customer --
Mary cannot be satisfied
The solution for customer --
John is [customer-orderNo/1,box-color/white,
      diskController-type/dc1,ioBoard-type/io1,
      processor-type/p3]
```

The flow is finished at step 10. We find that John's special requirements include "case", so processes c, d, e and f are triggered and finished. On the other hand, Mary's special requirements include "case" and "box", so processes c, d, e, f and g are triggered. But as Mary's special requirement for box-color/black can not be found in the entity database, the solution can not be provided to Mary. The flow stops at step "10" in example 4 because the termination "step" defined by the user is "10".

## 5 FWFL Workflow Engine

### 5.1 System Architecture

The system architecture of the FWFL workflow engine is shown in figure 4.

A BPM and an entity database are needed for this workflow engine. The BPM is composed of a *Process Specification*, a *Junction Specification* and an *Entity Specification*[8]. The definitions of the processes are stored in the *Process Specification*. It is written in FWFL which provides direct input to the workflow engine. If the BPM is changed, the definitions of the processes written in FWFL is changed correspondingly. Because of the direct mapping, it does not require much effort to deal with these changes. The purpose of the *Junction Specification* is to record the logical connections of the BPM; it also follows the FWFL. The *Entity Specification* stores the

---

[8]An entity is a data stored in the entity database.

definitions of the entities and events. Event occurrences and entity occurrences are stored in the entity database.

The user needs to determine how many steps that the flow needs to run. The flow may stop at a step which is stipulated by the user. After accepting the input data, the workflow engine starts running the processes based on the definitions of the processes and the BPM. It checks the triggered events first and triggers all the processes whose trigger conditions are satisfied. The logical meanings of the junctions direct the flow.

When processes are triggered, instances of those processes are created and executed. In each tick[9], all the possible instances are run within the given time until they are finished. Sometimes some instances may not be executed and finished in the current tick; the workflow engine keeps them until the next tick. The engine rechecks all remaining instances and runs them again in the next tick. This procedure loops until the flow reaches the step defined by the user at the beginning or until it reaches the "END" of the BPM. The time and flow state are updated in each tick, thus the user can know the time point and flow state in each step.

BPM 2002 Market Milestone Report[Group, 2002] classifies workflow processes into three categories "*process-to-process*", "*person-to-process*", and "*person-to-person*". The FWFL workflow engine implemented in this research captures formal descriptions of the processes and provides a structured method to capture the business process. In addition to handling "person-to-person" or "process-to-process" workflow processes, it also provides interaction between the user and the workflow system. As well it can deal with "person-to-process" workflow processes because it has some exception handling so the user can choose when he/she is asked for input data from outside the workflow system. In most situations, however, the workflow still works automatically.

There are two types of interaction in the FWFL workflow system that deal with the "person-to-process" workflow process. First, the workflow engine asks for new values (or updated values) from the user. In this case, the flow may stop and wait for input from the user. Second, the workflow engine may also detect conflicting actions which happen when different instances try to deal with the same entity data simultaneously. Warning messages are provide to the user, and the system asks for a decision.

Figure 5 shows the flowchart of the FWFL workflow engine which is implemented and based on this workflow meta-interpreter.

Table 6 shows the summary of the predicates used in the workflow system (meta-interpreter).

## 5.2 State Transition and Dynamic Behaviors

Once a workflow is implemented, we need to monitor its progress. We can do this by checking the status of a workflow [Georgakopoulos *et al.*, 1995]. The FWFL workflow system records the flow state using the predicate $flow\_state(Fstate, T)$ which indicates the flow state at time
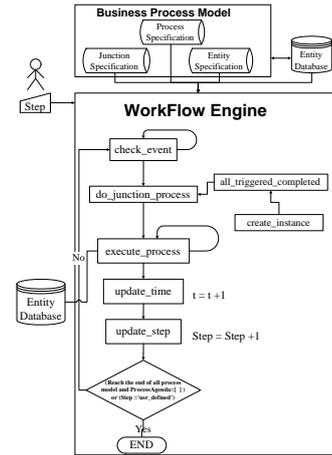
Figure 5: Flowchart of the FWFL Workflow Engine

| Predicate Name | Purpose |
|---|---|
| execute_flow | Control the whole process flow |
| check_event | Check the new event |
| do_junction_process | Execute the junctions of the BPM |
| execute_process | Execute the instances of the processes |
| update_time | Update the time point |
| update_step | Update the step counter |
| flow_state | Indicate the flow state at time T |
| check_mstate | Check the model state |

Table 6: Main Predicates of the FWFL Workflow Meta-Interpreter

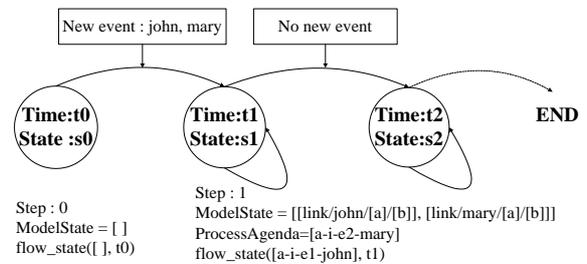T. Figure 6 shows an example of the state transition of the workflow system.



Figure 6: State Transition of the FWFL Workflow Engine

In figure 6, suppose we have a simple BPM:

junk(m1,start,[],[a]).
junk(m1,link,[a],[b]).
junk(m1,end,[b],[]).

and two trigger events at time point t1. What will be the state transition? In this case, the initial state is "flow_state([],t0)". The model state indicates the instances of the BPM for all trigger events. At the initial time, no trigger event occurs, so no instances of the BPM are created. The initial model state

is "[]". When two events – "customer-John" and "customer-Mary" are triggered, two instances of the model are created:

[start/john/[]/[a],link/john/[a]/[b]],
[start/mary/[]/[a],link/mary/[a]/[b]] [10]

After the instances of the model are created, the flow starts to run. The instances of the first process for each model may also be created and executed as long as the preconditions are satisfied. In this example, the precondition of process instance – "customer-John" is satisfied but the precondition of process instance – "customer-Mary" is not satisfied. Perhaps this is because the required input data is not available immediately. The instance of the process – "customer-Mary" is left in the execution queue and waits for data. These kinds of instance are run again at a later time point. The whole process flow of "customer-Mary" may stay at this stage until this instance is finished. On the other hand, the instance of the first process – "customer-John" – is executed and finished at time point t1 (Tick 1). The instances of the BPM change to

[link/john/[a]/[b]],[link/mary/[a]/[b]]

The flow state records the executed result – [[a1-i-(e1-john)], t1], and the process agenda stores the remaining instance – [a1-i-(e2-mary)] which may be executed later. The data created have been saved in the entity database, and that data may also be modified or removed later. The flow continues running, and the flow state changes at each stage until it reaches the end of the flow. There are two ways to reach the end of the process flow. First, when "ModelState" is an empty list and the "process agenda" is also an empty list, it indicates that the whole business process has executed completely. Second, the flow reaches a step defined by the user. In this case, the whole business process may or may not be finished.

## 5.3 Validation and Verification to Workflow System

Errors and warnings may happen in two situations: First, two or more instances try to deal with the same entity data. Second, some instances may be left in the execution queue for too long. This delay time duration is defined by the user using the predicate "*delay_time_duration(Time)*". Table 7 and table 8 contain detailed descriptions of this.

| Error Types | Description |
|---|---|
| Type I | Two or more different instances add, delete or update the same entity data at the same time. |
| Type II | One or more instances refer to entity data and others try to add, delete or update it. |

Table 7: Error Types of the FWFL Workflow Engine

When the workflow engine encounters these cases, the flow stops, a message is displayed and the system waits for a response from the user. Example 1 shows the message for a type I warning. It shows that the instance of process b-i-id(e1-John,e1-John) is already delayed for more than n- ticks (time). The system will show the warning message to ask

[10]The model instance of customer-Mary and customer-John.

| Warning Type | Description |
|---|---|
| Type I | An instance may not be executed for a long time. |

Table 8: Warning Type of the FWFL Workflow Engine

whether this delayed instance of the process should be kept or deleted. The user then makes a decision taking into account the impact of the decision. Serious problems may occur in this case. The flow may not be able to continue.

```
Example 1:
There is a delay in Process=> b-i-id(e1-John,e1-John)
Do you want to continue?(y/n)y.
Continue.....
Keep Process=>b-i-id(e1-John,e1-John)
There is a delay in Process=> b-i-id(e1-John,e1-John)
Do you want to continue?(y/n)n.
delete current process.....
```

## 6 Complexity of Business Process Models

The complexity of the BPM is discussed in this section. The complexity we calculate here is the complexity of the verification of the program using simulation/world state stepping techniques. The result shows that this program will be in practice need to carry out an exhaustive search to execute of all the possible permutation enactments of a business process model and find if there is any inconsistency. It also leads to a conclusion that "Or_split" junctions make a process model substantially more complicated as it allows rapid growth of different dynamic behaviours. In the following paragraphs we will show how we calculate the complexity of this program.

### 6.1 Complexity of a Single Model

In this section, we will calculate the complexity of a single model. Definition of the variables used in the following complexity analysis are:

- n: The number of branches from an And- or Or-Split junction.
- m: When two models (such as figure 1 (a)+(d), (a)+(c), (b)+(c), (b)+(d)) are connected, n is the number of branches from the first model and m is the number of branches from the second model.

An "Or-Or" model is used to illustrate how to compute the complexity. Suppose an "Or_split" junction has n branches. It is possible that all n branches, or only n-1 branches, are being triggered. In total, there are $n + (n-1) + (n-2) + \ldots + 1$ different possible triggered results. For each triggered process set, the execution sequence may also differ. The possible permutations of these processes should be considered at the same time.
The possible execution sequences of the "Or-Or" model are therefore:

$$\sum_{all\ permutations\ of\ these\ triggered\ processes} (each\ possible\ triggered\ result \times)$$

The second term in the product comes from:

$$\sum_{\substack{the\ second\ junction\ \times\ permutation\ of\ these\ processes\ \times \\ permutation\ of\ those\ remaining\ processes}} (the\ number\ of\ the\ finished\ processes\ before)$$

| Business process model types | Complexity |
|---|---|
| "And_Split" and "And_Joint" model ("And-And" model in figure 1 (a)+(c) ) | $O(n!)$ |
| "And_Split" and "Or_Joint" model ("And-Or" model in figure 1 (a)+(d) ) | $O(n \cdot n!)$ |
| "Or_split" and "And_Joint" model ("Or-And" model in figure 1 (b)+(c) ) | $\simeq O(n!)$ |
| "Or_split" and "Or_Joint" model ("Or-Or" model in figure 1 (b)+(d) ) | $\simeq O(n \cdot n!)$ |

Table 9: The Complexity of Different BPM Types

Based on the result computed by "Mathematica". We find that when "n=10", the value of formula 1 is $88 \times 10^7$ which is a very big number. The complexity of this kind of BPM is high and has a factorial rate of growth [Bard, 2001]. It is impossible to list all the possible execution sequences when a BPM becomes so complex. Table 9 lists the complexity of four different types of BPM.

As a result, we find the "Or-Or" model is the most complex model as it has the highest complexity. The "And-And", "And-Or" and "Or-And" are special cases of the "Or-Or" model. The "And-And" model case occurs when n processes are triggered at the same time and all the triggered processes must be finished before executing the following process of the next junction. The "And-Or" model case occurs when n processes are triggered at the same time and at least one of the triggered processes is finished at a later time point. The "Or-And" model is another special case when all triggered processes must be finished before executing the following process of the next junction. The following results are also found:

- The *Or_split* and *Or_Joint* have the greatest influence on the complexity. For instance, the complexity of "And-Or" is *n times* higher than "And-And".

- The complexity of these types of BPMs is very high, and it is difficult to carry out all of the possible execution paths.

## 6.2 Complexity of Combination Models

In this section, we try to combine some of the previous models and compute the complexity of them. They are categorised as:

1. A model finishing with an *And_Joint* junction.
2. A model finishing with an *Or_Joint* junction.

A model finishing with an *And_Joint* junction is considered first because it enables possible execution sequences of one independent model. In general, the complexity of this case is computed as:

$$\Pi(The\ complexity\ of\ the\ model\ finishing\ with\ an\ \text{"}And\_Joint\text{"}\ junction)$$

Models that finish with an *Or_Joint* have many possible execution results, especially when more than one Or_Joint is combined. These models may become very difficult to carry out.

A simple example – the combination of *And-Or* and *And-And* models – is used to describe the complexity. An assumption has been made to simplify this problem. We also remove the assumption to show the extent of the complexity. The assumption is that all triggered processes must be finished before the final process of each connected model (whereas in a real process model, an unfinished process may "propagate" and execute parallel to a process in the latter model).

Based on this assumption, the complexity of this problem is:
$\sum_{k=1}^{n} \frac{n!}{(n-k)!} \cdot (m+n-k)!\ (formula\ 2) \simeq O(m+n)!$[11]
As a result, formula 2 only provides the complexity under the assumption. If we relax the assumption, an "assumption term" has to be added. Then the result is:

$$The\ previous\ formula \\ +\ Those\ cases\ in\ which\ some\ processes \\ are\ executed\ after\ the\ final\ process$$

It will be larger than formula 2. Its complexity remains in $O(n!)$. The assumption does not have a big influence on the complexity.

As $n!$ has a factorial rate of growth which is bigger than polynomial rates, [Garey and Johnson, 1979] also shows that an NP-complete problem needs more than polynomial time to solve (i.e. it needs exponential time or greater). The growth rate of $n!$ is bigger than the growth rate of $k^n$, so we know that the business process problem is at least an NP-complete problem. [Chen-Burger, 2002b] indicates that workflow systems may be required to handle over 300 processes based on a real military BPM. The possible execution sequences of such a large model may be more than $60! = 8.3 \times 10^{81}$, if there are many different types of the junction in the BPM (i.e. if there is a total of 300 processes and there are 5 junctions, then each junction will have approximately 60 branches). So the number of possible results may be enormous if we do not consider the detailed semantics of the processes.

## 7 Other Related Work

In order to show the difference between FBPML + FWFL and other workflow process languages, the application of Petri nets to workflow management[van der Aalst, 1998] is compared in this section. The most significant difference between "FBPML + FWFL" and this research is that "FBPML + FWFL" has a formal business process modeling mechanism which separates the business and the implementation logic. Hence the workflow system is more flexibly reactive to a dynamic environment.

A Petri net that models a workflow process definition is called Workflow net (WF-net). Comparing the formal method (FBPML + FWFL) that we provide, and Petri net (WF-net), the following results have been found (the details of WF-net are not explained here, we focus only on the comparison between them):

- Because a Petri net is a process modeling technique, it focuses on workflow processes which are designed to handle cases (called instances in FWFL). However, it does not focus on resources, such as people, machines or organization units, which may be involved in the workflow system. The concept of "Role" is ingrained in FBPML+FWFL in order to represent resources and use them in the BPM.

---

[11]When m=0, this formula represents the complexity of the "And-Or" model.

- Tasks are modeled by transitions in Petri net, whereas in FBPML, activities are used to describe tasks. In Petri nets, cases are modeled by tokens, whereas in FBPML + FWFL, instances are used to describe cases.

- In Petri net, each condition is modeled by a "place" and it may represent a precondition or a postcondition of the process. In FBPML + FWFL, logical meanings of the junction and preconditions in the attributes of a process replace the usage of "place".

- To define a process and a BPM, FBPML + FWFL has a formal, declarative semantics. A process described in WF-net does not have clear definitions of its attributes.

- To represent different tokens (cases or instances), there are two ways to represent and list them in Petri net (WF-net): 1. Use different colors in a high-level Petri net[12]. 2. Use instances to describe them. The latter is the same in FBPML + FWFL, but instances in FBPML+FWFL are not listed in the BPM. Using FWFL, they are only described as instance occurrences inside the workflow engine.

- High-level Petri net has a time extension to describe the temporal behavior of the system. In FBPML + FWFL, "Precedence-Link" is used to indicate a temporal constraint between two processes. It is similar to Petri net.

- High-level Petri net provides a hierarchy construct called "subnet" which can be used to structure large processes. In FBPML + FWFL, a high-level process can be divided into two different types of low-level sub-processes (decomposition and alternation) representing a similar notion.

- Both Petri net and FBPML+FWFL have a "trigger" notion. The trigger condition is clearly defined as an attribute of a process in FWFL. The user may formally describe the trigger condition of this process. The declarative description in FWFL makes the trigger condition easy to understand.

  In Petri net (WF-net) the trigger concept is distinguished by different symbols and focuses on the enabler (Automatic, User, Message, Time) which triggers the process. It does not focus on the data or world state conditions. In FBPML + FWFL, trigger conditions are flexible; the user may define any trigger condition as long as he/she follows the language.

- Junction notions used in Petri net (WF-net) are modeled by ordinary transitions. Transitions are treated as control tasks. In FBPML + FWFL, each junction has its own formal definition. Logical meanings of *And_Split* and *And_Joint* are the same in FBPML+FWFL and Petri net (WF-net). However, in Petri net (WF-net) the "OR-split" is categorised into "implicit OR-split" and "explicit OR-split" (the same as "OR-join"). Although this may provide a clear definition of workflow modeling, it

makes notation more complex[13]. FBPML + FWFL does not distinguish this case so that the modeling language is easy to learn because it is more natural. Users do not need to remember specific meanings of notations.

- The WF-net is designed to require that there are no dangling tasks and/or conditions in WF-net. Every task (transition) and condition (place) should contribute to the processing of cases. The concept is similar to "irrelevant nodes" in our three-level framework.

- Triggers and workflow attributes are removed when analysing workflow in Petri net (WF-net). In our three-level framework, we also only consider trigger conditions at the level 3 – case study. The reason[14] is the same as that described in [van der Aalst, 1998]. In our framework, in order to actually simulate possible model behaviors, some semantics and details of the process are considered in the analysis. This is different from Petri net (WF-net) analysis in which all verification and validation are done at a graphical level; although it has more formal definitions for verification than ours.

## 8  Conclusions

In this research, we try to bridge the gap between *Enterprise Modelings* (EMs) and *Software Systems* in order to provide support where EMs are used as a part of KM initiative. This gap exists primarily between the capabilities for gathering and presenting knowledge, and the capability for performing semantic-based automatic manipulation of this knowledge. Formality needs to be introduced to the informal or semi-formal enterprise modeling paradigm to provide precision and enable automatic support. A workflow system is built to bridge this gap, and allow domain knowledge to be checked for consistency and correctness during enterprise modeling. The following conclusions are made:

- FBPML is a merger of two standardised process modeling languages: IDEF3 and PSL. The benefit of merging the two languages is that the former has graphic notation but lacks formal process conceptualisation, whereas the latter provides formal process theory without any visualisation. Although the two languages are not equal, their core concepts overlap. Such core concepts are included in FBPML, and are carefully disposed so that the consistency of FBPML is maintained.

- The graphical notation used in FBPML is intended to make it easier for those unfamiliar with predicate logic to describe models in the language. Although we have not conducted extensive empirical evaluations of the FBPML graphical language, it is very similar in style to other graphical process modelling languages that have achieved widespread use. Our aim here is to conform

---

[12]A Petri net extended with color, time and hierarchy is called high-level Petri net.

[13]The author also says that there is no compelling need to distinguish between implicit and explicit OR-joins, but there is for 'OR-split'.

[14]The reason is that it is impossible to model the behavior of the environment completely.

to currently accepted modelling practices. The graphical language in FBPML, however, translates automatically to a predicate logic description that supports both a declarative reading (helpful in checking the logical consistency of the model) and multiple operational readings (allowing different forms of enactment engines to execute a process model by interpreting the model description). As usual, we make our enactment engines generic for all forms of FBPML model so that we can freely change the declarative description without needing to alter the enactment engines.

- The workflow meta-interpreter is based on FBPML + FWFL. It accepts input specifications and executes the BPM directly. It may be implemented as a "person-to-process" workflow system so that some validation and exception handling may be confirmed by the user. This makes it a more useful tool for the user due to this flexibility. Most of the time, flow is executed automatically. Thus we have a flexible way to execute a business process flow in a dynamic environment.

- Our three level-framework provides a thorough test which is useful in analysing a BPM. At level 1, after carrying out syntactic critiques, the appropriate topology for a complex BPM is mapped out. At level 2, the details of the process are considered, and features from this topology are inferred. Because of this explicitness due to inference of the model, some impossible execution results are eliminated after performing semantic critiques. Problem size is therefore reduced. Level 3 corroborates a BPM written in FBPML + FWFL with respect to a special domain. Thus, the three-level framework is useful for these four reasons:

  1. A BPM is always complicated.
  2. Details of processes must be considered when executing a process.
  3. Errors and warnings also need to be checked to insure the accuracy of a model.
  4. A case study for testing a BPM is a good way to simulate possible execution results, and to make a model and workflow system more accurate. This detects some possible malfunctions before the model runs online which saves time and cost.

  This three-level framework is used to analyse a business process model, and it does not have restrictions on the language used to describe a model.

- The complexity of a BPM has, at least, a factorial rate of growth. The program used to verify a process model needs to carry out an exhaustive search to execute of all the possible enactments of a business process model. This will be hard. That is why we provide a three-level framework to analyse a BPM in a more organised way.

### Acknowledgements

## References

[Bard, 2001] Jonathan F. Bard. Classification of integer programming problems. www.me.utexas.edu/b̃ard/NP-Complete.doc, 2001.

[Chen-Burger *et al.*, 2002] Yun-Heh Chen-Burger, Austin Tate, and Dave Robertson. Enterprise modelling: A declarative approach for fbpml. *European Conference of Artificial Intelligence, Knowledge Management and Organisational Memories Workshop*, 2002.

[Chen-Burger, 2002a] Yun-Heh Chen-Burger. Akt business process model akt project, akt web site (http://www.aktors.org), 2002. http://www.aiai.ed.ac.uk/project/akt/work/jessicas/pc-config/onto-mode/top-level.html.

[Chen-Burger, 2002b] Yun-Heh Chen-Burger. Sharing and checking organisation knowledge. *Knowledge Management and Organizational Memories*, ISBN 0-7923-7659-5, July 2002. Publisher:Kluwer Academic Publishers, Boston Hardbound, Editors: Rose Dieng-Kuntz, Nada Matta.

[Garey and Johnson, 1979] M. Garey and D. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman And Company, 1979. ISBN: 0716710455.

[Georgakopoulos *et al.*, 1995] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases, 3, 119-153*, 1995.

[Group, 2002] Delphi Group. Bpm 2002: Market milestone report. *Web site: www.delphigroup.com/ coverage/ bpm_webservices.htm*, February 2002.

[Mayer *et al.*, 1995] Richard Mayer, Christopher Menzel, Michael Painter, Paula Witte, Thomas Blinn, and Benjamin Perakath. *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*. Knowledge Based Systems Inc. (KBSI), September 1995. http://www.idef.com/overviews/idef3.htm.

[Sadiq and Orlowska, 1996] Wasim Sadiq and Maria E. Orlowska. Modeling and verification of workflow graphs. No. 386, Department of Computer Science, The University of Queensland, Qld 4072 Australia, November 1996.

[Schlenoff *et al.*, 1997] Craig Schlenoff, Amy Knutilla, and Steven Ray. Proceedings of the process specification language (psl) roundtable. *NISTIR 6081, National Institute of Standards and Technology, Gaithersburg, MD*, 1997. http://www.nist.gov/psl/.

[Schlenoff *et al.*, 2000] C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, and J. Lee. The process specification language (psl): Overview and version 1.0 specification. *ISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD (2000)*, 2000. http://www.nist.gov/psl/.

[ter Hofstede *et al.*, 1996] A.H.M. ter Hofstede, M.E. Orlowska, and J. Rajapakse. Verification problems in conceptual workflow specifications. *International Conference on Conceptual Modeling / the Entity Relationship Approach*, 1996.

[van der Aalst, 1998] W.M.P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers, 8(1):21-66*, 1998.

[Yeh and Young, 1991] Wei Jen Yeh and Michal Young. Compositional reachability analysis using process algebra. *Symposium on Testing, Analysis, and Verification*, 1991. Software Engineering Research Center, Department of Computer Sciences, Purdue University West Lafayette, IN 47907.